2013-09

Fingerprinting reverse proxies using timing analysis of TCP flows

Weant, Matthew S.

Monterey, California: Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

FINGERPRINTING REVERSE PROXIES USING TIMING ANALYSIS OF TCP FLOWS

by

Matthew S. Weant

September 2013

Thesis Co-Advisors:	Geoffrey Xie
	Robert Beverly
Second Reader:	Justin P. Rohrer

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
09-27-2013	Master's Thesis	2012-08-15—2013-09-27

4. TITLE AND SUBTITLE

FINGERPRINTING REVERSE PROXIES USING TIMING ANALYSIS OF TCP FLOWS

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Matthew S. Weant

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

13. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.IRB Protocol Number: N/A

14. ABSTRACT

Reverse proxy servers are valuable assets to defend outside hosts from seeing the internal network structure upon which the reverse proxy is serving. They are frequently used to protect valuable files, systems, and internal users from external users while still providing services to outside hosts. Another aspect of reverse proxies is that they can be installed remotely by malicious actors onto compromised machines in order to service malicious content while masking where the content is truly hosted. Reverse proxies interact over the HyperText Transfer Protocol (HTTP), which is delivered via the Transmission Control Protocol (TCP). TCP flows provide various details regarding connections between an end host and a server. One such detail is the timestamp of each packet delivery. Concurrent timestamps may be used to calculate round trip times with some scrutiny. Previous work in timing analysis suggests that active HTTP probes to servers can be analyzed at the originating host in order to classify servers as reverse proxies or otherwise. We collect TCP session data from a variety of global vantage points, actively probing a list of servers with a goal of developing an effective classifier to discern whether each server is a reverse proxy or not based on the timing of packet round trip times.

15. SUBJECT TERMS

Active Measurement, Timing Analysis, Reverse Proxy, Fingerprinting

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER *(include area code)*
Unclassified	Unclassified	Unclassified	UU	101	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

FINGERPRINTING REVERSE PROXIES USING TIMING ANALYSIS OF TCP FLOWS

Matthew S. Weant
Captain, United States Marine Corps
B.S., Computer Science, United States Naval Academy, 2006

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2013

Author: Matthew S. Weant

Approved by: Geoffrey Xie
Thesis Co-Advisor

Robert Beverly
Thesis Co-Advisor

Justin P. Rohrer
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Reverse proxy servers are valuable assets to defend outside hosts from seeing the internal network structure upon which the reverse proxy is serving. They are frequently used to protect valuable files, systems, and internal users from external users while still providing services to outside hosts. Another aspect of reverse proxies is that they can be installed remotely by malicious actors onto compromised machines in order to service malicious content while masking where the content is truly hosted. Reverse proxies interact over the HyperText Transfer Protocol (HTTP), which is delivered via the Transmission Control Protocol (TCP). TCP flows provide various details regarding connections between an end host and a server. One such detail is the timestamp of each packet delivery. Concurrent timestamps may be used to calculate round trip times with some scrutiny. Previous work in timing analysis suggests that active HTTP probes to servers can be analyzed at the originating host in order to classify servers as reverse proxies or otherwise. We collect TCP session data from a variety of global vantage points, actively probing a list of servers with a goal of developing an effective classifier to discern whether each server is a reverse proxy or not based on the timing of packet round trip times.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

3WHS Three-way Handshake
ARPANET Advanced Research Projects Agency Network
AS Autonomous System
C&C Command and Control
CDN Content Distribution Network
CSV Comma-Separated Values
DHCP Dynamic Host Configuration Protocol
DNS Domain Name Server or Domain Name Service
DOD Department of Defense
FSM Finite State Machine
HTML Hypertext Markup Language
HTTP Hypertext Transfer Protocol
HTTPS Hypertext Transfer Protocol Secure
ICMP Internet Control Message Protocol
IG Information Grid
IP Internet Protocol
ISP Internet Service Provider
IT Information Technology
MITM Man-in-the-middle
NAT Network Address Translation
NPS Naval Postgraduate School
OTT One-way Transit Time
PET Privacy Enhancing Technology
PHP Hypertext Preprocessor
P2P] Peer-to-Peer
RFC Request For Comments
RPC Remote Procedure Call
RTT Round Trip Time
SRTT Smoothed Round Trip Time
TCP Transmission Control Protocol
TSopt TCP Timestamp Option
USMC United States Marine Corps
USN United States Navy
XML EXtensible Markup Language

THIS PAGE INTENTIONALLY LEFT BLANK

Executive Summary

A reverse proxy server handles incoming network requests by maintaining two connections; the connection from the requesting host external to the network, and a second connection with the data storage host internal to the network. The requesting host only sees the traffic between itself and the reverse proxy server, and the internally networked, data storage machine only sees the traffic between itself and the reverse proxy server. This hidden traffic concept supports network access control, security protection through obfuscation, and performance boosts at the Internet facing access points for handling thousands of incoming requests.

Reverse proxies are used for a variety of reasons. One such way they are used is for security and information hiding. Nefariously, botnet controllers use numerous ways to mask their origin and implementation in order to maintain operations, whether the task is spam, denial-of-service attacks, or otherwise. Additionally, home users and large businesses alike implement security measures to protect the data that they serve. Reverse proxies hide the network assets on the non-Internet facing routes. This is helpful in two major ways; providing a single source of traffic control for monitoring network activity and hiding the internal network infrastructure from the requesting host(s). Another way reverse proxies are used is for performance boosts. As a performance boost, reverse proxies are employed by Content Distribution Networks (CDNs) such as Akamai or web-hosting services to support fast delivery of web-based content for online companies by routing host requests to their reverse proxy servers.

Webservers such as Apache and Nginx provide high performance, reverse proxy solutions. It is easy to see that highly trafficked Internet sites can increase their client throughput by utilizing reverse proxy servers that increase their potential for sales or advertising. From a more malicious viewpoint, botnets pose a serious threat to network bandwidth, home users, companies, corporations, and governments. Millions of dollars are spent annually to deter botnet operations such as spam and denial-of-service attacks from harming Internet users and service providers. The ability to identify machines that are a part of a botnet or botnets can help deter or neutralize botnet attacks, either by blocking network connectivity between the infected host and the webserver or by taking the infected host offline through legal means. As stated previously, reverse proxy servers are also helpful for maintaining network infrastructure from a managerial perspective. Researchers studying network edges may find it helpful to know that a single destination address is actually a reverse proxy, handling HTTP requests for any number of content providers. Probing techniques and research results could be tailored to handle reverse

proxies once they are identified as such. If my advisors and I are able to correlate timing analysis with reverse proxy servers to a high enough level of confidence, we could distribute the method to security groups or research organizations. The concept could be used to perform self-observational security penetration tests, analyzing CDNs and DNS lookup correlation(s), or to map out networks for military use, similar to Intelligence Preparation of the Battlespace (IPB) applied to network warfare.

Acknowledgements

I'd like to thank Professor Geoffrey Xie and Assistant Professor Robert Beverly for all their keen understanding, help, and patience with me over the past year. I would not have been able to manage my projects and research as efficiently without their support. Many thanks to Professor Xie for keeping me on track through his informal style and positive attitude to ensure I met my self-imposed deadlines. Professor Xie was outstanding at breaking down large projects into smaller, more manageable tasks that reduced stress and kept learning and researching fun. Assistant Professor Beverly helped me routinely with understanding various networking topics of interest, programming, PlanetLab deployments, and thorough feedback on our work all along the way. I am a better researcher because of both of them. I am also grateful to Dr. Rohrer who aided me in my explanations by providing me thorough feedback in his areas of expertise.

I also want to thank my cohort friends and classmates at school (Jeremy Martin, Danny Rhame, Chad McDowell, and Tim Garcia) for keeping me on my toes. Their optimism and sense of responsibility strengthened similar traits in me. I appreciate each of them. I specifically want to show my appreciation to Tim Garcia and Rob Yee. Tim was my partner for a lot of class projects and was always willing to help me solve problems by listening and providing candid feedback. Rob was my first project partner that led to my thesis research, and put up with a lot of early weekend mornings to get work done outside of school. Thanks to everyone that helped!

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1: INTRODUCTION

As the number of malicious actions increases daily on the Internet, so must defenses for companies, governments, and even home users. Since the early 2000s, there has been a significant expansion in the number of network security services to combat malicious network operations. Those who conduct malicious acts on the Internet have learned that they must defend their own platforms as well. Reverse proxies can be used by malicious actors as well as trusted agents. Trusted agents and security professionals would prefer to deter malicious activities before they occur. Network researchers may also wish to map the topology of a network. The ability to identify system characteristics and network setups could contribute to identifying malicious origins, leading to a better network understanding for security professionals and researchers alike. Therefore it is important, from both the defense and offense perspectives, to characterize the behaviors of server types. More specifically, reverse proxies provide several functionalities that can be used effectively by trusted agents and malicious actors similarly. Reverse proxy servers are assets to defend outside hosts from seeing the internal network structure upon which the reverse proxy is serving. Using timing analysis of TCP and HTTP flows, which is the protocol upon which reverse proxies operate, there is a wealth of timing information that can translate into usable intelligence for detecting the use of reverse proxies by a network domain.

1.1 Problem Statement

The ability to check data authenticity is very important given that malicious network attacks are occurring more often than once a second on the Internet [1]. How do Internet users recognize when they are connecting to an authentic content provider and not some malicious back-end server? Unsuspecting users could be vulnerable to man-in-the-middle (MITM) attacks or web-based attacks [2] against web browsers and browser plug-ins. There is no straightforward answer to this question because of the numerous technicalities introduced by network protocols and structure. Domain names are a vulnerability, for example, because individually they provide no stamp or insight into the trustworthiness of the server to which they direct users, but they are a pivotal functionality of web browsing. In order to increase performance amongst other reasons, content and other service providers use proxies to control the flow of network traffic and return content to customers. A widely used type of proxy server used by webservers and Content Distribution Networks (CDNs) is the reverse proxy.

A reverse proxy server handles incoming network requests by maintaining two connections: the connection from the requesting host external to the network, and a second connection with the data storage host internal to the network. The requesting host only sees the traffic between itself and the reverse proxy server, and the internally networked, data storage machine only sees the traffic between itself and the reverse proxy server (See Figure 2.3). This hidden traffic concept supports network access control, security protection through obfuscation, and performance boosts at the Internet facing access points for handling thousands of incoming requests.

Reverse proxies are used primarily for security and information hiding. Nefariously, botnet controllers use numerous ways to mask their origin and implementation in order to maintain operations, whether the task is spam, denial-of-service attacks, or otherwise [3], [4], [5]. Additionally, home users and large businesses alike implement security measures to protect the data that they serve. Reverse proxies hide the network assets on the non-Internet facing routes. This information hiding is helpful in two major ways; providing a single source of traffic control for monitoring network activity and hiding the internal network infrastructure from the requesting host(s).

Another way reverse proxies are used is for performance gains. Reverse proxies are employed by CDNs such as Akamai to support fast delivery of web-based content for online companies by routing host requests to their reverse proxy servers. Reverse proxies are able to cache HTTP requests that help increase throughput and lower latency for web clients [6], [7]. Additionally, Internet and web service providers can use reverse proxies to handle incoming requests across web hosting farms [7] handling multitudes of web sites and their content. Webservers such as Apache and Nginx provide high performance, reverse proxy solutions to service thousands of requests per second. It is not difficult to see that highly trafficked Internet sites can increase their client throughput by utilizing reverse proxy servers which increases their potential for sales or advertising.

The ability to reliably identify reverse proxies is valuable to better understand a network topology as well as identify possible vector origins for malicious actors. Botnets pose a serious threat to network bandwidth, home users, companies, corporations, and governments. Millions of dollars are spent annually to deter botnet operations such as spam and denial-of-service attacks from harming Internet users and service providers [8], [9] as well as cleaning up the mess that botnets leave behind. The ability to identify machines that are a part of a botnet or botnets can help deter or neutralize botnet attacks, either by blocking network connectivity between

the infected host and the webserver or by taking the infected host offline through legal means. Addtionally, researchers studying network topologies may find it helpful to know that a single destination address is actually a reverse proxy, handling numerous requests for any number of content providers. Probing techniques and research results could be tailored to handle reverse proxies once they are identified as such.

There are some methods that can be used to detect reverse proxy servers. Each technique currently known requires at least one controlled variable in addition to the standard network traffic generated. For example, there exists a system [10] that can detect reverse proxies but it requires the use of the EXtensible Markup Language (XML). This is difficult in practice because not all websites employ applets and XML. We desire to fingerprint all reverse proxies with minimal limitations. No publicly known techniques have shown a process to detect or classify reverse proxies using only the TCP traffic from a remote host, specifically the TCP Round Trip Times (RTTs).

1.2 Research Description and Hypothesis

Our hypothesis is that we can identify, or fingerprint, reverse proxy servers by analyzing the TCP flows between remote hosts under our control and end hosts by comparing the RTTs for the three-way handshake to the RTTs for the HTTP traffic. Reverse proxy servers should show an increased RTT for some of the HTTP packets since the reverse proxy server has to forward the external requests to an internal machine. This may not be the case with caching, since reverse proxies are also used to reduce load on internal network webservers. However, we hypothesize that the malicious hosts running webservers are unlikely to implement caching with their exploited hosts and they often need to fetch some dynamic content for targeted attacks, even with caching turned on. We pose the following primary research questions:

1. Is the timing analysis of TCP flow RTTs sufficient to classify webservers as reverse proxies with reasonable accuracy?
2. Will RTTs from the initial handshake RTT to establish a TCP connection be equivalent for data RTTs for a direct connection?
3. Can RTT ratios of the GET RTT over the 3WHS RTT for multiple TCP connections to a single server be used as a threshold for classifying those servers as direct connections or reverse proxies?
4. How reliable are existing TCP flow tools when identifying and calculating RTTs?

1.3 Organization

Chapter 1 is an introduction to our thesis research. It presents high-level reasoning for pursuing the identification of reverse proxy servers. Additionally, Chapter 1 covers the organization and structure for the thesis paper in order to frame the problem and solution method.

Chapter 2 provides the baseline for the research. There are several sections describing the history of key concepts for the study and contributing elements to our research. This chapter motivates the purpose and focus of the research topic, and sets up the reader to fully understand the methodology and approach in follow on chapters.

Chapter 3 uses the concepts and fundamental knowledge provided in Chapter 2 as a starting point for creating our fingerprinting strategy. We look at several test cases to analyze how we can collect network traffic for detailed RTT analysis. Lastly, we develop a methodology for collecting flows and their corresponding RTTs.

The results of our data collection and our analysis are presented in Chapter 4. We look at data organization and website groups first. Secondly, we analyze the results of each website group with comparison between the groups. The reverse proxy results are provided as a transition into fingerprinting characteristics that we use to perform a selection of case studies for individual websites.

Chapter 5 includes the conclusions of the research and future work. We describe the strengths and weaknesses, contributions, and future work of our research.

CHAPTER 2:
BACKGROUND AND RELATED WORK

The Internet has evolved from a simple communication sharing scheme between universities to a global network of systems. The network communication path established between a client and a server on the Internet is directly related to the network architecture which is never controlled by a single entity. The architecture is a shared responsibility amongst network administrators, autonomous system owners, organizations, communications companies, and so forth. One of the methods by which network hosts may gain some control over their network path to a server is through the use of proxies. Proxies also provide a level of anonymity for clients and servers by serving as an intermediate node in a network path on behalf of the requesting host. In regards to the Internet, web content delivered via proxies can be monitored, hidden, or rerouted to provide security for either the requesting host or the servicing host. In some cases, the service that proxies provide is used for malicious reasons. We want to identify the systems that are being used for malicious proxy use by performing active measurements, conducting network traffic analysis, and implementing a statistical fingerprinting technique. In order to accomplish proxy identification, we first need to understand the previous work.

2.1 Origins and Fundamental Knowledge

The original use of the Internet was designed for ease of sharing information between remote locations [11]. As the Internet spread globally, privacy and security became more desirable for online communication, banking, e-commerce, and data storage to name a few. When new technology for privacy and security is introduced for public use, there are groups of people who want to break those secure methods. The interests for breaking privacy and security protocols and programs are not necssarily malicious. Proving that a tool or system is unbreakable is valuable information. One such approach to breaking privacy is the ability to fingerprint a target, whether that be a person, system, or otherwise, based on an identifiable set of characteristics. The set of characteristics in which we are interested are those affiliated with reverse proxies. Reverse proxies are commonly associated with web proxies, services that handle incoming HTTP requests to a web server. As such, they are connection-oriented services that utilize the HTTP and TCP to handle traffic between external web clients and internal data servers. The time it takes a web server directly connected to the open Internet to handle HTTP requests should be less time than a reverse proxy should take since the reverse proxy has to handle the

front end connection from the web client as well as the back end connection the appropriate data server. We will describe the various types of proxies and the types of connections that can occur to a web server. There are several methods to make web servers more efficient, such as caching. If a reverse proxy uses caching, the time to respond should be reduced since the back end data server connection would not be required. We look at the types of caching before identifying similar network traffic analysis methodologies to break privacy tools. Lastly, we present several fingerprinting methodologies. Fingerprinting looks at the characteristics of connections and network traffic analysis to help us identify reverse proxies.

2.1.1 Connection Types

There are three primary ways that a host can establish a connection via Internet services using TCP and HTTP. These three ways include a direct connection, a redirected connection which is essentially a sequence of two or more direct connections, or a proxied connection.

Figure 2.1: Direct Connection - The host and the server communicate directly with each other

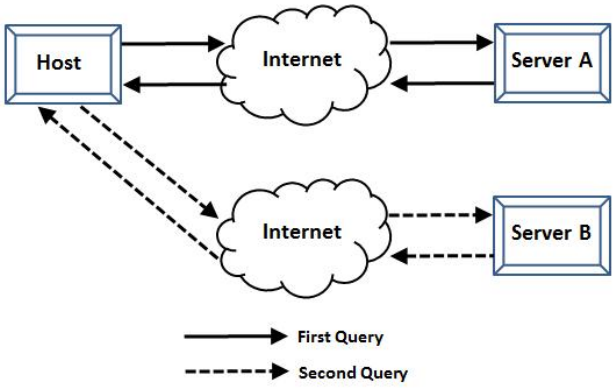


Figure 2.2: Redirected Connection - The host connects to Server A, then gets redirected to Server B

Direct connections are not common in the present Internet [12]. Redirected connections are much more common for a variety of reasons. Webpage images, advertisements, or other portions of a website may be hosted on multiple webservers. These webservers are not necessarily in the same location. The requesting host is then required to make more than one direct connection to retrieve the various portions of a single webpage. Though the retrieval may seem like a single web request, the host is actually making several connections to gather all the fractional

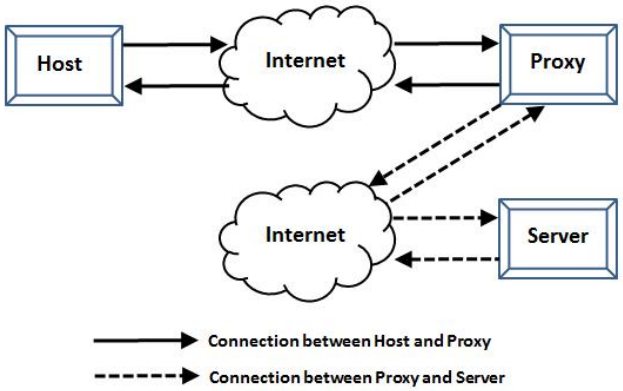


Figure 2.3: Proxied Connection - The proxy sends and receives traffic for the host

parts to make the whole. Anonymity may also be a factor in connections, which is the case when using proxy services. Proxied connections can be established in multiple ways, which will be covered in the next section.

2.1.2 Proxies

A proxy is defined as an agent or authority who is authorized to act on behalf of someone else [13]. In the case of networks, we use proxy servers. A proxy server is defined as a server that acts as an intermediary for requests from clients seeking resources from other servers [14]. There are various types of proxy servers that are used to facilitate network traffic on the Internet. Proxy servers are helpful in providing anonymity to Internet users by handling traffic requests as the originating host. Proxy servers can enable a user to maintain privacy when requesting content from a server or hide the identity of a network attacker, shape traffic based on incoming and outgoing connection information, and they can also protect precious resources for a company or web service from external threats.

There are several types of proxies regularly used on the Internet. The most widely-used type is known as a forwarding proxy (Figure 2.4). A forwarding proxy handles the origin host's request by forwarding the request to the end host on behalf of the requesting host. The end host sees the original request coming from the proxy, so the end host responds with the appropriate message to the proxy. The proxy then forwards the end host's response to the originating host. In this case, the originating host maintains anonymity because it only connects and communicates with the forwarding proxy server. The forwarding proxy server maintains two connections; one connection between itself and the origin host and a second connection between itself and the destination host, or server. The destination host only sees the proxy as the originating host,

and does not know anything about the actual originating host. A secondary proxy type is the open proxy (Figure 2.5), which is merely a forwarding proxy that handles forwarded traffic at a middlepoint in the Internet.

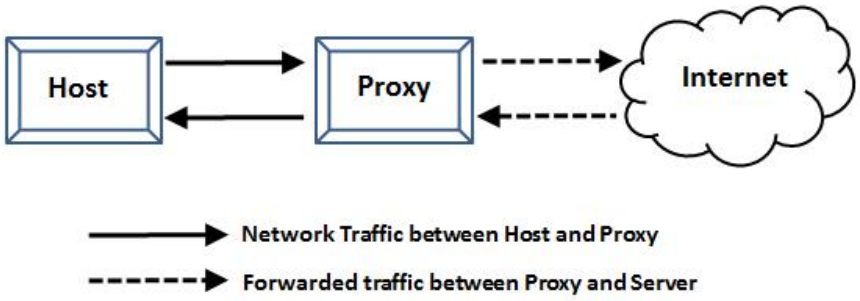


Figure 2.4: Forwarding Proxy - Forwards traffic from an internal network request to a destination on the Internet

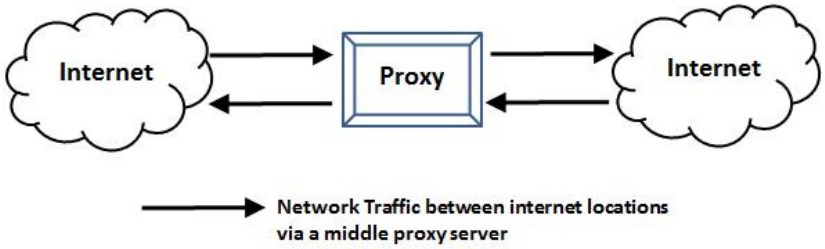


Figure 2.5: Open Proxy - forwards traffic from two hosts on the Internet, but maintains no connections between the two

Another common form of proxy server is the reverse proxy (Figure 2.6), which is occasionally referred to as a web cache [15] or an HTTP accelerator [16]. A reverse proxy takes requests external to the network it is serving, and forwards the external requests to the appropriate internal host or server. The external host likely does not know that its request is being transferred beyond the reverse proxy server, therefore providing anonymity to the internal network. The internal network may not know the external host that submitted the request because it only sees the request coming from the reverse proxy server. The reverse proxy server maintains the two connections just like the forwarding proxy, but instead of providing anonymity to the requesting host, a reverse proxy server provides anonymity to the end host or hosts. The reverse proxy server is generally an application that often works closely with the internal network firewall [17] so that external requests can only access the reverse proxy address.

There are several other forms of proxies, such as caching and DNS proxies to name a few. These are more functionally specific proxy servers and are not pivotal to understanding this thesis topic. The only item of note is that reverse proxies can have a dual role as a caching proxy if it is customized to do so. This would make the connection appear more like a direct connection from the origin host's point of view.

There are a variety of software solutions to handle proxy requirements. Most web browsers have network options to set forwarding proxy addresses in order to route HTTP traffic for client workstations. Some internal networks force this requirement to monitor intranet activities by forcing all connected workstations to route their traffic through an administratively controlled server before reaching the open Internet. This is mostly for handling outgoing traffic. Conversely, reverse proxy software solutions are common to handle network loads and protect internal network servers. A few common examples of these reverse proxy programs are Nginx [18], Apache [19], and Squid [20]. Each of these software packages exhibit similar packet handling techniques when routing HTTP requests. Malicious users may create their own reverse proxy server solutions to reduce their externally visible footprint.

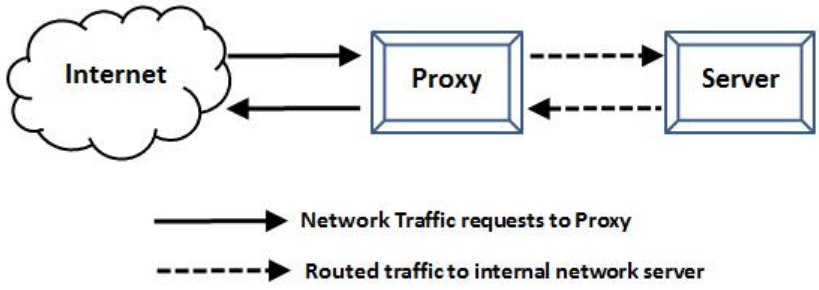


Figure 2.6: Reverse Proxy - Receives Internet requests and forwards the request to the appropriate internal server and back to the requesting Internet host

2.1.3 Web Browsing and Caching

Web browsing is the most common use of HTTP. The general concept is that an originating host, or web client, establishes a connection with an end host, or web server. The web server has data content made available via a shared filesystem, typically a web server which allows connections via ports 80, 443, or 8080. These ports are the standards, but web servers can be configured to accept connections on different ports. Once the two hosts are connected via a three-way handshake using TCP packets, the web client submits a request to the web server in the form of an HTTP message. This request can be a GET, HEAD, POST, PUT, DELETE,

TRACE, or CONNECT [21]. The most commonly used requests are the GET, POST, and PUT requests. Once the web server receives the request, it acknowledges the request and proceeds to send TCP packets with data comprised of HTML code as the base object of the request. It is common to have additional objects within a single web page such as embedded javascript applets, images, or dynamic content such as advertising services. For example, a user submits a web request using a web browser. Once the HTTP data is received by the client machine, the web browser will notice embedded links within the HTML of the web page to various elements. Each additional element will initiate another request from the web browser, resulting in numerous GET requests originating from the web client machine. The take away here is that the HTML content is the base information required to render a web site to a requesting web client. The additional elements are usually links to specific pieces of content for the website, and are standalone network requests when considering the network traffic from a web client to a web server. Sometimes the extra content objects reside on the same web server as the HTML. In cases such as this, additional connections are not necessary. The browser window will display the content of the data transfer as it becomes available [12].

Web browsing requests may be directed to Content Distribution Networks (CDNs) instead of the actual web server through Domain Name Service (DNS) request handling. A CDN is a large network of servers deployed across multiple networks in several data centers, often geographically distributed [22]. CDNs are employed on the Internet to reduce network load on the original web server, reduce load on the Internet as a whole, and provide faster response times to edge hosts that typically make the web requests. The way that CDNs provide current web content is by replicating their customers' data throughout their CDN data centers [15].

CDN distribution nodes keep track of the most current web server content. When an update occurs, the distribution node pushes the new material to each of the data centers. These data centers may operate behind reverse proxy servers at the Internet edge. In this case, reverse proxies would handle the incoming requests and pull the data from the data servers or use the cached data at the proxy server. CDN servers rely on edge host DNS requests for web pages to serve web content. They intercept web page requests from edge hosts and redirect the edge hosts to their CDN data centers. Edge hosts can bypass CDN services by identifying the origin web server's Internet Protocol (IP) address and connecting directly to that address vice using the DNS service.

CDNs are an interesting case of web browsing connections. They are globally positioned to

provide data content to end users. Each CDN stores thousands of web page files as a service to companies [23] willing to pay for this performance boost. The types of companies that typically request CDN support are those that serve thousands or more clients across multiple geographic regions. Typical companies include commercial online stores, news agencies, and corporations. CDNs must pay Internet service providers to provide Internet connectivity. There are also non-commercial examples of CDNs. Peer-to-peer CDNs are used in distributing files across the Internet. Hosts may "opt-in" to such a CDN in order to download files. The concept is that the originating data provider takes less of the network load as participating users increases. Each user's host machine contributes to sharing the file once the file is downloaded. This type of CDN is not as pertinent to web services such as HTTP, and will not be discussed further.

Another way to enhance web browsing performance is through web caching. This is different from CDNs in that web content can be cached at the edge host making the request or any server along a network path between the web client and the intended destination, the web server. Some sources will define a web cache as a form of proxy [15], though web caching can occur at the edge host as well. Web caching is much like memory system caching on a computer [16] in that the cache stores anticipated requests to speed up response time. Caching at a reverse proxy server will affect packet round trip times. If the reverse proxy has the most up-to-date content, then it can serve the data immediately instead of handling a connection to the backend server. Round trip times would appear to be those of a direct connection instead of a reverse proxy.

An interesting caching mechanism that closely resembles a reverse proxy is that of the transparent cache [7]. Transparent caches (see Figure 2.7) intercept port 80 web requests at the switch and/or router and serve web content using policy-based redirection to one or more back-end web cache servers or cache clusters. The main difference between a reverse proxy and a transparent cache is that a reverse proxy is software based whereas a transparent cache is implemented through hardware and routing policies. An additional difference is that transparent caching can be done throughout a network, and is not solely implemented near the end of the network path like a reverse proxy server. Overall, caching is implemented heavily throughout the Internet. Conversely, we expect that caching on compromised hosts is highly unlikely because it would significantly increase the footprint of the malicious user which is deemed undesireable.

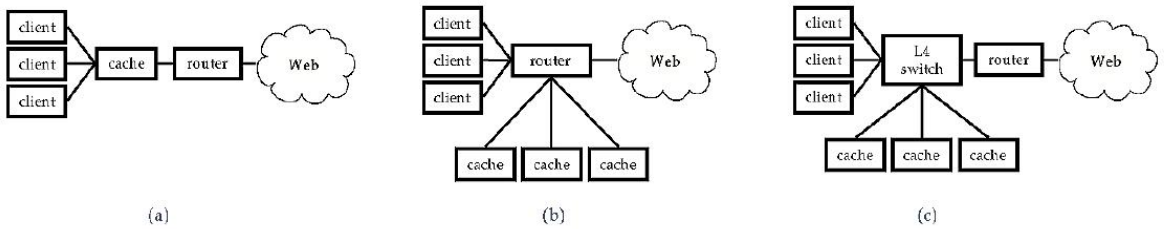


Figure 2.7: (a) Standalone cache (forwarding proxy), (b) Router - Transparent cache, and (c) Switch - Transparent cache

2.2 Related Work

There are several fields of study that focus on identification, classification, and analysis of network characteristics. Fingerprinting is an area of research that attempts to match an identifiable characteristic to a host, server, or user. Traffic Analysis attempts to understand network and user behaviors by examining the movement of data across systems. Timing analysis utilizes network timing data from delays, packet round trip times from host to host, as well as clock skews to build a classification mechanism. These methods are reviewed below as related work to this study.

2.2.1 Fingerprinting

The technique or techniques used to identify unique features of a program, service, or system is defined as fingerprinting. Fingerprinting techniques are typically classified as active, passive, or semi-passive [24]. Kohno et al. describes these techniques as the following. Passive fingerprinting techniques are where the fingerprinter, a measurer, attacker, or observer, is able to obsere the traffic from the device, or fingerprintee, that the attacker wishes to fingerprint. An active fingerprinting technique is where the fingerprinter must have the ability to initiate connections to the fingerprintee. A semi-passive technique is defined as a situation where, after the fingerprintee initiates a connection, the fingerprinter has the ability to interact with the fingerprintee over that connection. An example provided states the fingerprinter as a website with which the device is communicating, or is an Internet Service Provider (ISP) in the middle capable of modifying packets en route [24].

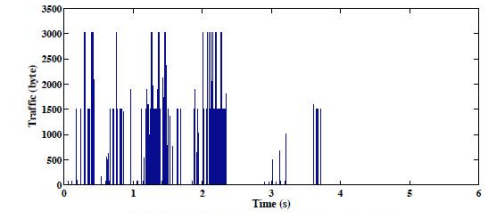
The Internet is a vast network. Researchers will rarely have direct access to perform passive or even semi-passive measurements. Because of this, we desire active methodologies that we can use to gain knowledge where there previously was none. We achieve this goal through iso-

lating unique characteristics of our identified target and formulating algorithmic steps by which we fingerprint systems, normally via remote means. There are a variety of known methods for fingerprinting systems. Several of the approaches that are pertinent to this study include clock skew fingerprinting, website fingerprinting, reverse proxy fingerprinting using the Extensible Markup Language (XML), statistical network traffic fingerprinting of botnets, and timing channel fingerprints.

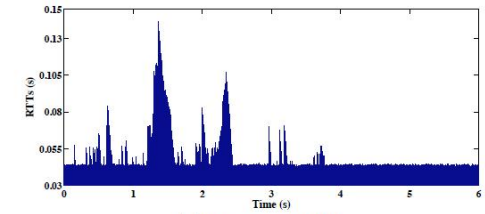
There are a multitude of ways to identify operating systems remotely. A recent study found that system clock skews can be used to fingerprint not only an operating system but uniquely identify individual systems with microscopic timing deviations in hardware [24]. An important discovery in this research was that computer chips have uniquely identifiable clock skews that are discernable from practically anywhere using active measurements over either ICMP or TCP. They were able to validate that system clocks and TSopt clocks were highly varied across large sets of systems. This is important to note since our active measurements for this thesis will be focused on our web client system clock times instead of TSopt values. The time stamps that are used in packet captures are pulled from the operating system kernel [25]. Therefore, time stamps for incoming and outgoing packets are all based off of the web client's operating system and not exact times in reality. A second important takeaway from this research was that, due to the high amount of variability in network delays referenced in Vern Paxon's paper [26], some sort of linear programming or smoothing mathematics are required to apply fingerprinting measurements with an acceptable level of accuracy. This thesis research will use a statistical threshold to handle the timing fingerprint used against web servers for reverse proxy identification.

There are a few notable techniques for fingerprinting websites based on network traffic analysis. One such way builds on Hintz's groundwork which simply counts the downloaded files and file sizes for individual websites and compares those two indicators against SafeWeb user traffic passively [27]. The second iteration of research performed by Gong et al. [28] actively studied spikes in traffic patterns using side channels and the queues on links coming from edge network hosts. The authors monitored the encrypted network traffic on the incoming DSL link while actively pinging the target host. Based off of the latencies resulting from queueing, they were able to fingerprint the target host websites visited by using Time Warping Distance (TWD) as a similarity metric (Figure 2.8 [28]). Graph (a) represents the number of bytes transferred over time when requesting the website Yahoo.com. Graph (b) represents the observed RTTs from pings sent to the target host. Graph (c) is the processed version of Graph (b), bringing

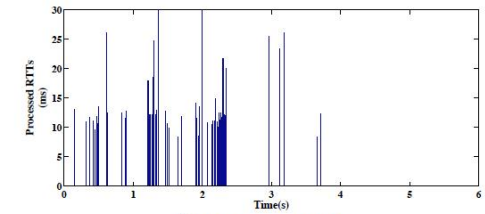
out the distinct spikes in RTT behavior that can then be compared to graph (a) for matching target host website activities. The RTTs of their pings indicated differences from the norm, in this case queueing, and could be compared to their known data sets of website traffic patterns to identify, or fingerprint, the edge host network activity. Likewise, we will use RTT differences from our hypothesized expectations to differentiate web server types.



(a) HTTP trace of Yahoo.com



(b) Observed RTTs



(c) Processed RTTs

Figure 2.8: Real traffic on a DSL vs. probe RTTs

The second methodology for website fingerprinting utilized a multinomial Naïve-Bayes classifier to subvert Privacy Enhancing Technologies (PETs), such as Tor, VPNs, and JonDonym, using IP packet frequency distributions [29]. The Naïve-Bayes classifier must assume independence among the variables, which is undesireable. This type of classifier has a successful history in website fingerprinting research. Though the use of the classifier is very different, we will employ a Naïve-Bayes classifier to identify reverse proxies once we have an established timing threshold. Similar to the previous website fingerprinting research, the attacker must have a database of known and/or anticipated websites to fingerprint the target user's traffic. Our methodology requires a list of websites that we use to generate traffic captures, but does not require any former traffic knowledge to fingerprint servers. Once we establish the threshold time

expectations for a reverse proxy server versus a typical Internet-facing web server, we expect to be able to identify any IP address serving web content as a regular web server or a reverse proxy.

United States Patent Number US 7,272,642 B2 [10] covers the detection of reverse proxies and establishing connections to the hidden servers via tunneling by analyzing differences in affinity identifiers in network traffic. Affinity identifiers represent specific portions of a physical or virtual device. Examples include MAC addresses, IP addresses, virtual network identifiers, and so on. The specific affinity identifiers that they reference in their work focus on differences between destination IP addresses of active code, or java applets, and the web server.

Active code segments within a web page are not capable of being cached, yet are linked within the web page. The links to the active code segments within a web page are observable, and can be compared to the web server address for differences. If it is determined that there is a difference in destination IP address, the research presents a methodology to establish a tunnel directly to the webserver by crafting specific HTTP requests utilizing the outlying affinity identifier(s), effectively bypassing and fingerprinting any reverse proxy in the process. This detection patent will only work for web servers that handle active code segments within the web content, though it could be combined with our work to have more thorough fingerprinting results.

Peer-to-Peer botnets are currently a more common architecture because they are harder to take down, more resilient to host gain and loss, and stealthier in their activities. Zhang et al. [30] describe a methodology in an enterprise network where they monitor all network traffic at edge nodes to identify P2P activity. The fingerprinting focus is on the C&C communication patterns. They use a phased approach, which is helpful in managing the fingerprinting process.

The first phase is to reduce the large amounts of captured network traffic to only that which could potentially be related to P2P communications. From this reduced network traffic they use statistics to isolate flows related to P2P communications and identify P2P clients within the network. The second phase they implement a detection system that analyzes traffic generated by the previously identified P2P clients and classifies the clients into legitimate P2P clients or bots. An interesting theory that they utilized with great effect was that bots within a botnet operate the same way throughout the botnet. We could attempt to take this theory for our research, assuming that reverse proxy signatures for similar botnets or C&C managers would remain consistent, leading to a wider view of reverse proxy operations within a botnet.

One of the most interesting fingerprinting studies is the work done by Shebaro et al. [31]. They proposed an active method to leave a fingerprint on machines hosting illegal services that, in conjunction with timestamps from hidden service log files in those hosting machines, could be used to validate illegal activity in a criminal case. The anonymity approach used by Tor hosting machines was to have a web server application such as Apache [19] using pseudodomain names. The researchers used HTTP GET requests from controlled hosts through the Tor network to a controlled Tor hosting machine. They were able to monitor hidden service logging by Apache and Tor as well as control the rate that GET requests were submitted through the Tor network. They sent GET requests every 5 seconds over 55 hours to identify delay distributions through the Tor network since the Tor routes were changed every 10 minutes. They built Reed-Solomon codes that were basically code words for GET request submissions at 1 bit per minute for each bit of the code word to the hosting machine. This timestamp of requests, which was handled in conjunction with delay distributions to ensure a noticable fingerprint was embedded, was identifiable on the hosting machine in the hidden service log files. In our research we do not expect to leave any fingerprints of our activity on the target machines.

2.2.2 Traffic Analysis

Traffic analysis studies have been performed since the Internet began with the ARPANET at the Network Measurement Center of University of California-Los Angeles [11], [32]. Traffic data records the time and duration of communications, and the analysis of this data can be used for a variety of purposes. Several of the purposes are important because the timing of network traffic affects host and server performance, delays imparted onto all users of a network, as well as the quality of some services such as streaming media. We need to understand network impacts on packet round trip times such as variance and latencies, packet dynamics, and the various ways timing analysis can be affected and used to glean understanding and knowledge. Traffic analysis has military and law enforcement roots [32]. It was used to gain intelligence against adversaries and criminals. The root goal of traffic analysis is to understand what is going on in an area where the amount of data is overwhelming and non-intuitive. We will first look at network dynamics that pertain to traffic and timing analysis, then address several important analysis research topics that pertain to our current research.

Vern Paxson did a study on packet dynamics in the late 1990s that underlined the difficulties with larger scale network measurements [33]. He first describes that analyzing packets using TCP requires a way to gather timestamp values for the sending and receiving of those packets, which TCP does not always provide using TSopt. We rely on packet filtering tools

and system clocks to provide timing values for packet activities. These filtering tools, tcpdump and wireshark for example, may improperly label times or miss timestamps entirely because of "measurement drops." Furthermore, TCP packets can be sent over a short duration for small TCP flows or for hours in cases where there are large numbers of packets transferred in large files. These irregularities complicate our ability to do correlational and frequency-domain analysis. This point is addressed in many studies, including this thesis project, by using a number of tests deemed statistically sufficient to cover the normal cases with guaranteed outliers. A high enough number of test cases can average or smooth out the results in order to do correlational and/or frequency-domain analysis.

Out-of-order packet delivery is prevalent in the Internet due to the close proximity of data packet transmission which is affected by queueing rates, route flutter where packet routes change over time, and assumed packet loss combined with duplicate packet transmissions. More specifically, data packets are more often reordered than acks because they are frequently sent closer together [33]. Route flutter is defined as the "fluttering" of network routes changing which affects the timing and arrival of packets. Route flutter is assessed as being site dependent. Some websites exhibit higher amounts of route flutter and out-of-order packet arrivals than other sites. This point contributes to our reasoning for performing a large number of HTTP requests from a single host to a single target which we will highlight in Chapter 3. Though the routes may change and timing data may be inconsistent for some flows, the averages should be telling.

Packet loss in the late 1990s was calculated between 2.7% and 5.2%. The larger percentage was assessed to be partially due to a bigger transfer window that allowed for more data in flight. This makes sense because queues would build up faster and have a higher likelihood for dropping packets. Internet bandwidth has increased significantly since this study. As such, it is expected that packet loss rates are even lower than reported here. A note of interest regarding packet delay is that one-way transit times (OTTs) are often not well approximated by dividing RTTs in half, and that variations in the paths' OTTs are often assymetric [33]. Lastly, packet and timing compression can occur because of bottlenecks along transit routes. This could potentially sway timing results with RTTs given a small sample size. As previously mentioned, we will perform a large number of probes in order to analyze timing characteristics without being blinded by timing or packet compressions.

Martin et al. [34] examined the different delay times on the Internet focusing mainly on

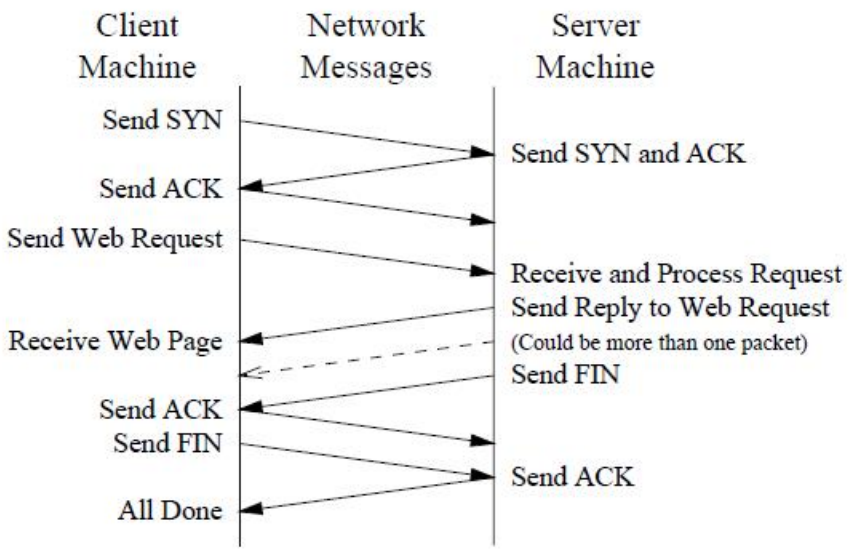


Figure 2.9: Simple HTTP session

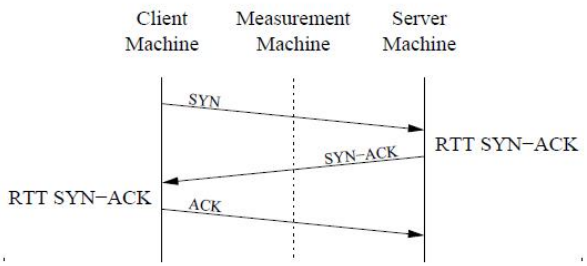


Figure 2.10: SYN / ACK RTT

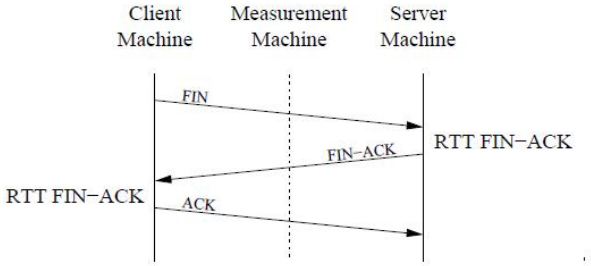


Figure 2.11: FIN / ACK RTT

HTTP traffic and server-side delays. Both delays are primary factors in our research. They analyze HTTP flows from both directions, identifying message types (SYN, ACK, FIN, etc.) and assessing packet timeouts or packet loss. They define a typical HTTP flow (Figure 2.9 [34]), and provide examples of RTT values for the typical HTTP flow (Figures 2.10, 2.11, 2.12, 2.13 [34]). Figure 2.9 [34] does not reflect what is seen in reality. The difference between real web requests and those examined in this study, as shown in Figure 2.9 [34], is that real web requests ACK the web request first then send follow-on packets with data. The web request ACK does not contain any data.

The authors also acknowledge the variety of RTTs in a simple HTTP flow. The first distinct RTT is the three-way handshake (3WHS) which is comprised of a SYN from the client to the server followed by a SYN-ACK from the server to the client. The web request from the client

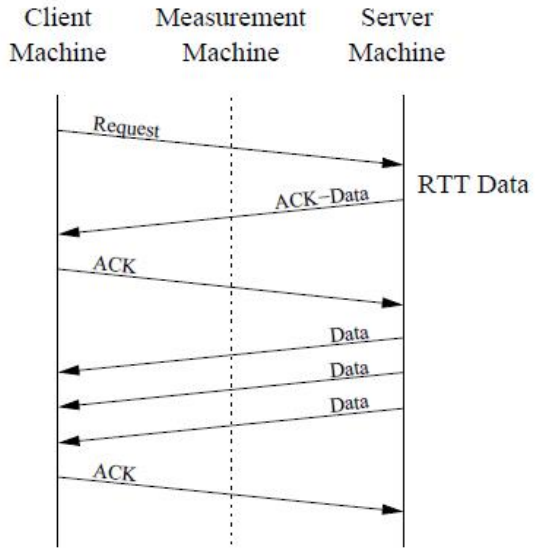


Figure 2.12: Data Packet Timing Example

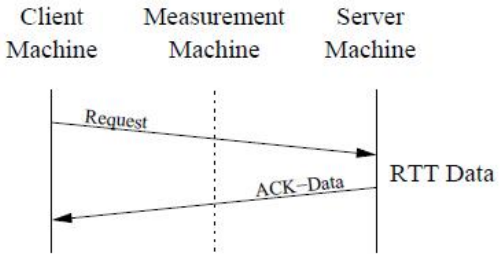


Figure 2.13: Data Packet RTT

to the server and the server response can also be classified as an RTT. For the purposes of our research, we will define this web request RTT in Chapter 3. It will not be consistent with this research or most RTT estimation research in that we include multiple packets for identificaiton. The estimated percentage of lost and duplicate packets were consistent with Paxson's findings (roughly 2% to 8%) [33].

Martin et al. propose to use the minimum RTT as a measure of latency within a connection. However, they do recognize that taking a minimum RTT value is very sensitive to errors. If the route changes in the middle of a flow, the measurement could be miscalculated. This risk was mitigated by executing longer lasting flows in the hours or close to it. A key takeaway in queueing delay is that the two directional links in a connection can be unambiguously aligned so that no replies occur out of order. For the purposes of network measurement accuracy, a few milliseconds of variance in precision is sufficient given that the size of most delays are in the tens or hundreds of milliseconds. Lastly, it was concluded that network delays were the dominant effect on overall delay within HTTP flows. This is very important for our research since we ascertain that the network delays will cause timing values to "give away" the presence of a reverse proxy.

Karn's algorithm [35] defines how RTTs should be measured when handling packet retransmissions for network traffic shaping. Karn's algorithm is very important to understand because

most previous work with RTTs utilize this definition. It states that "When an acknolwedgement arrives for a packet that has been sent more than once (i.e., retransmitted at least once), ignore any round-trip measurement based on this packet, thus avoiding the retransmission ambiguity problem. In addition, the backed-off RTO for this packet is kept for the next packet. Only when it (or a succeeding packet) is acknowledged without an intervening retransmission will the RTO be recalculated from SRTT" [35]. This definition of RTTs is used in most current traffic flow analysis tools such as TCPTrace, tstat, and others. This presents challenges when attempting to use current tools for our work. We will have to build our own definition of RTTs not only to include RTTs that would previously have been ignored but also to omit some RTTs that would not be indicative of a reverse proxy.

An important study on variability in round trip times [36] within TCP connections showed that RTTs within connections vary widely. This impacts our research because we cannot simply average round trip times and estimate small differences as valid identification measurements. There must be a distinct difference in timing values for the round trip times, and the variability in the round trip times must be examined closely to increase classification accuracy. This same paper notes that ACK packets may be delayed in some connections to every other packet, which is an optional TCP feature [37]. Should this occur, RTT averages for comparison in our study would be skewed. Along this same line of thinking, TCP slow start will cause variance with packet RTTs since more data will be sent. In addition to using Karn's algorithm in this RTT variability study, Aikat et al. added additional tests to reduce RTT samples even more. We will diverge from both of these RTT sampling methodologies in that we primarily care about the three-way handshake RTT and the time difference between the client's GET request and the first data packet response from the webserver.

Vern Paxson examined the clock skews and clock adjustments from client and server vantage points to see how common they were between separate hosts [26]. He discovered that they are indeed common, and that using the network time protocol (NTP) does not resolve these errors. His discovery suggests that, when measuring packet features and delays from two or more endpoints, clocks are rarely ever synchronized. Since they will be off in their timestamps, it is erroneous in many cases to correlate client and server clock values in measurement calculations. Times from one host should be isolated and addressed only with times from that host. He also describes clock offsets. System clocks will be offset to some degree from the time established by national standards. He defines skew as the frequency difference between the clock and national standards, a simple inverse formula from the offset value. The remaining portion of his

work here focuses on an algorithm for identifying clock adjustments and the variability in clock characteristics. We will not be "cross-pollinating" our timing values between server network traffic samples and client traffic samples. In fact, our client traffic samples are our main focus. The majority of our network measurements in this research project will be solely from the client's perspective. Future work may address multiple vantage points, and for this reason this is included as a factor in measurement calibrations.

The final piece of related work we wish to discuss revolves around the precision of timestamps for recorded network traffic [38]. Jörg Micheel et al. recognize the differences in latency since Paxson's packet dynamics work in the late 1990s as a growing issue in network measurement studies. Smaller latencies over the Internet mean that the once acceptable 10ms resolution of system clocks is no longer an option to get valuable test results. They propose the implementation of a hardware-driven timestamp using DAG measurement cards [39]. They were able to increase timestamping precision by at least an order of magnitude, but software solutions remain the norm throughout most of the world. In future experiments, these hardware-based clocks might be imperative to gain valuable timing differences in higher-speed networks. Though the current state of the Internet provides high-speed connections with low latency values, the RTT values are not small enough yet to be washed away by typical system clock time resolutions.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
REVERSE PROXY FINGERPRINTING METHODOLOGY

3.1 Overview

We first discuss our initial study and analysis of RTT values for a direct HTTP connection with a web server as well as a connection through a reverse proxy. This motivates the deeper review of packet dynamics within HTTP requests. The differences in TCP flows between standard web servers and reverse proxy webservers are discussed in detail in order to highlight unique characteristics. These characteristics are used in creating an algorithmic approach to step through a single TCP flow to identify the values for the 3WHS and GET RTTs. The need for a multi-flow algorithm is presented for large-scale data parsing of pcap files. The method we use for data collection and reasoning for our choices of web sites upon which we test our algorithm is presented to motivate our implementation in the next chapter.

3.2 Analysis of Basic Concept

As a precursor to this thesis research we did a preliminary study to determine whether or not round trip times could differentiate between direct connections to web servers and from those to reverse proxy servers. We built a simple web page that was less than 1000 bytes in size in order to be delivered in a single data packet, which is the best case possible since there is no server-side processing. We hosted this website through the GoDaddy webservices [40] at `http://www.4550testing.com/`, which hosts all websites on an internal network unless the website has signed up for a dedicated IP address. In the first phase of our experiment we scripted HTTP requests and network traffic captures from the client machine located in Monterey, CA, directly to the hosted website whose server was in New Jersey. In the second phase of the experiment we ran the same script but sent the requests to a reverse proxy server built off of Apache [19] located in Monterey, CA, which handled backend traffic to our hosted website.

Since this was our first attempt in identifying RTTs, we designed a method to give us some control in the packet transfer between the client and the server in order to identify the GET request. We built a simple webpage in html that was smaller than the maximum segment size (MSS) for a single TCP packet. Using this data size gave us the ability to see the entirety

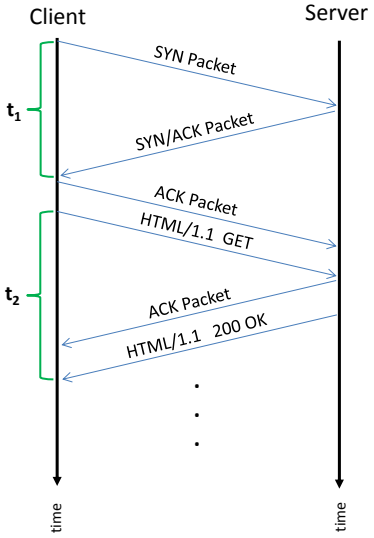


Figure 3.1: Filtering Method to Identify GET RTT - Direct Connection

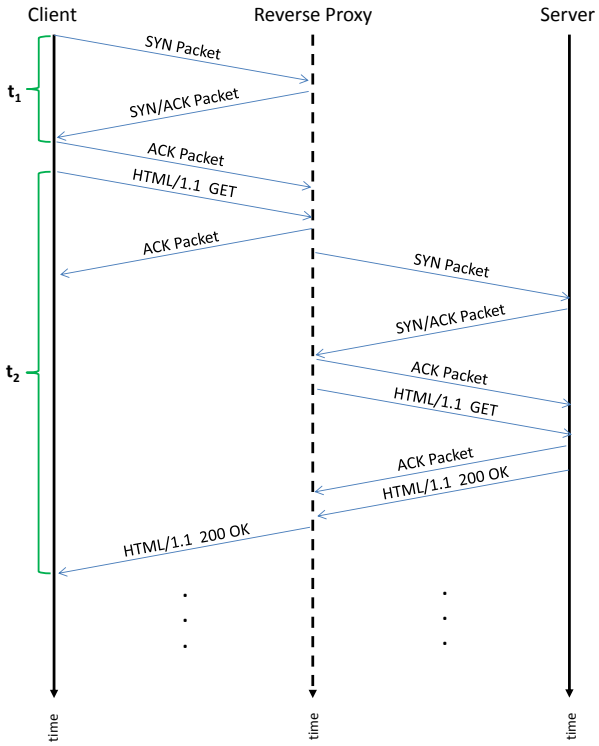


Figure 3.2: Filtering Method to Identify GET RTT - Reverse Proxy

of the webpage delivery in a single packet once the GET request was acknowledged by the server. Our methodology for pulling the GET RTT was to save the packet data in text format using Wireshark's [41] command-line version known as Tshark. We then used linux regular expression tools such as sed, awk, and grep to look for the HTTP GET packet and the HTTP/1.1 200 OK packet. Since the webpage was delivered in a single packet, we could use these filters to find the timestamps to calculate the GET request RTT (Figures 3.1 and 3.2) where t_1 and t_2 denote the 3WHS and GET RTTs respectively. We did not know that there was an ACK packet following the GET request during this experiment, but identified the GET RTT based on our expectation that the GET response packet would contain data and the HTTP 200 OK message. Figures 3.1 and 3.2 reflect actual packet transitions so that the reader may understand why our experiment approach worked and to motivate our GET RTT analysis in a later section. This method was sufficient for our small, controlled experiment. We knew we would need to adapt our methodology when experimenting with real world web sites because of variable-sized data transfers, interacting with different network infrastructure, variable distances to proxied

webservers, and so forth.

We inspected all of the network traffic capture files for the direct connections (Figure 3.3) as well as those for the reverse proxy (Figure 3.4). In the direct connections, the 3WHS and data RTTs were nearly identical. However, the reverse proxy RTTs were significantly different as we expected. The additional time for the reverse proxy to establish a backend connection to the content-hosting server added extra latency for the data packets. The results from this simple experiment strengthen our hypothesis that it may be feasible to identify reverse proxy servers with a TCP timing analysis.

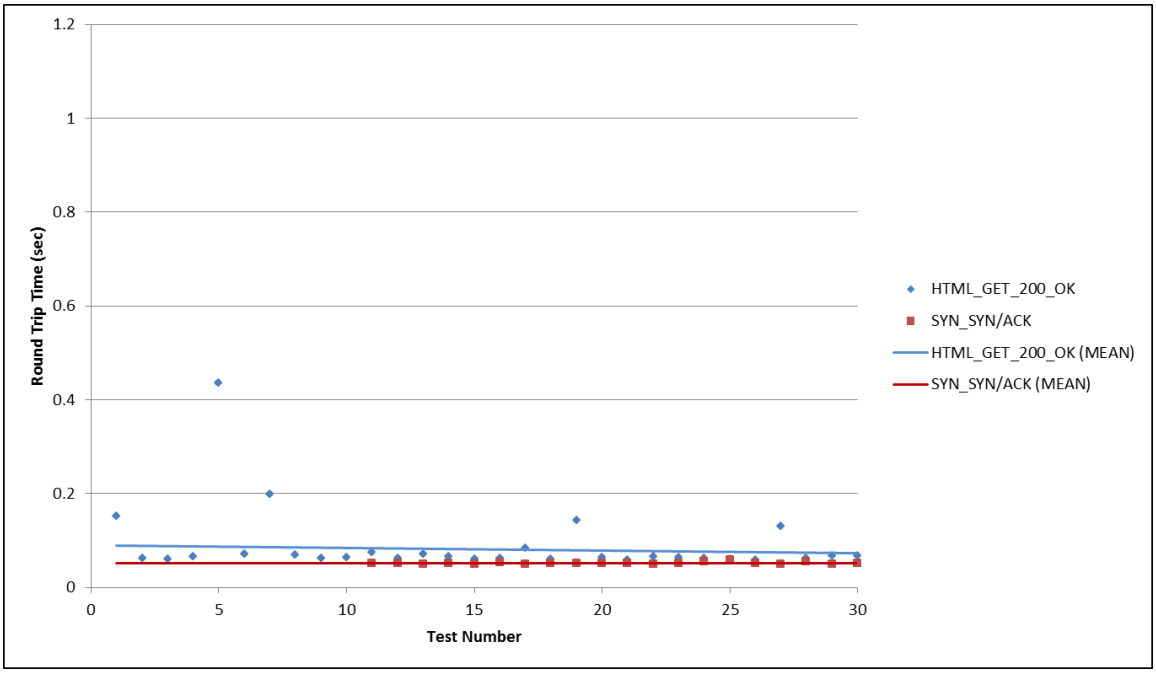


Figure 3.3: Direct Connection RTTs

3.3 Analysis of Packet Dynamics within HTTP Requests

We need to understand how packets are exchanged in typical TCP flows in order to identify proper RTT values. We define a flow as the connection setup, data transfer, and connection termination. Our first examination is to look at packet behaviors within direct connections. Following these tests, we will implement the same requests but perform them through reverse proxy services. To ensure consistency across reverse proxy services, we run the same tests against Apache version 2.2.22 (Ubuntu), Squid version 3.1.20, and nginX version 1.2.1 individually. We controlled the client and the reverse proxy, so we were able to capture the network traffic from both vantage points.

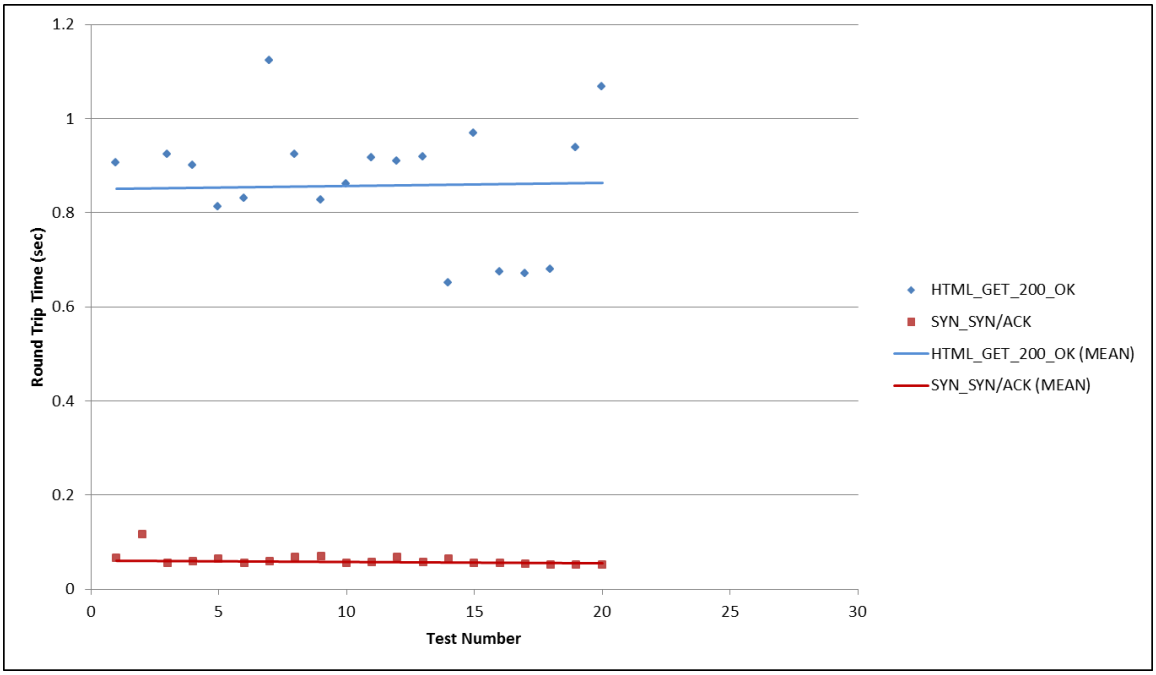


Figure 3.4: Reverse Proxy Connection RTTs

3.3.1 Base Cases

The basic direct connection test was performed using Apache web server configured as a regular webserver (Figure 3.5). The client workstation used Mozilla Firefox as the web browser. The client made a direct connection using the webserver's IP address and the resulting TCP traffic was collected at the client and reverse proxy server vantage points. We looked at the traffic generated from both vantage points to look for differences.

We expect that the GET request following the 3WHS will be replied to with an ACK and data in a single packet, but this is not the case. The GET request receives an ACK response with no data, followed shortly thereafter with a data packet. TCP slow start begins thereafter. The 3WHS is denoted as t_1 and t_2 is the time between the HTTP GET request and the first data packet. We refer to the ACK TCP packet to the HTTP GET request as the GET ACK packet. The GET ACK is a function of TCP that requires acknowledgement of packets that contain data, which in this case contains the HTTP request in the TCP data section [42] from the client to the server. We define the GET RTT as the time between the HTTP GET request and the first packet that contains data.

We used the same setup for the basic reverse proxy connection test (Figure 3.6) except that we configured Apache to operate as a reverse proxy to GoDaddy [40]. We capture the network

data from both the client and reverse proxy vantage points, similarly comparing the data as we did in the basic direct connection. The 3WHS RTT and time between the HTTP GET request and the GET ACK packet were nearly the same RTTs as those we observed in the direct connection test. The difference in time for the first data packet to arrive is significantly different. The reverse proxy has to establish a connection with the back end server (GoDaddy [40]) before it can provide data to the client. Production reverse proxy servers may maintain connections with back end servers, negating the requirement for a second 3WHS. The data in production servers still has to be processed and submitted, however, so a larger RTT for the GET RTT will still exist.

Once several data packets are received by the reverse proxy, the data is transferred to the client. This back and forth process continues similarly to the direct connection network traffic. The client network capture shows no noticeable differences between the reverse proxy and direct connection except for the drastic increase in t_2. We draw the timing diagrams in Figs 3.5 and 3.6 based on the network traffic we observe in both pcap files to highlight the timing difference. Though not highlighted in Figures 3.5 and 3.6 the connection tear down has the same behavior as the 3WHS. The difference between Figures 3.5 and 3.6 and Figures 3.1 and 3.2 is that we recognized the HTTP GET request ACK packet instead of focusing on HTTP messages.

3.3.2 Test Case Examples

Referring to Figure 3.5, we can see that the initial packet exchange is as expected for the three-way handshake. We conduct the same direct connection and reverse proxy tests using two separate web browsers, an HTTP request tool, and three web server programs.

Using Apache as a reverse proxy as we did in the base case shows identical network traffic results between Internet Explorer, Mozilla Firefox, and WGET (Figure 3.7) because TCP handling in the kernel is common to all three. The time to establish a 3WHS(t_1) was only 2.2 percent of the GET RTT (t_2). The time between the HTTP GET request and the GET ACK packet was 0.018131 seconds, which is a similar RTT to the 3WHS RTT and is calculated using Karn's algorithm [35]. The 3WHS RTT was roughly 45 percent of the time of the GET RTT as defined by Karn's algorithm. This difference is minimal. When we look at the time required for the first data packet to arrive, we can easily see the variation in timing between the RTTs which is indicative of the reverse proxy.

NginX and Squid had similar behaviors. They were handled differently by the back end

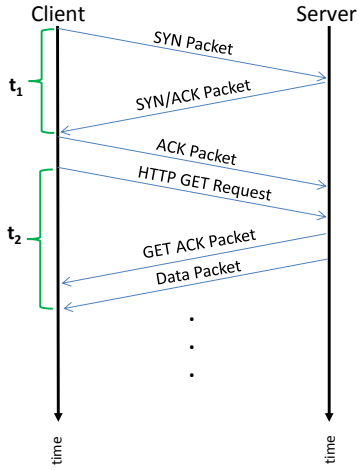


Figure 3.5: Direct Connection RTTs

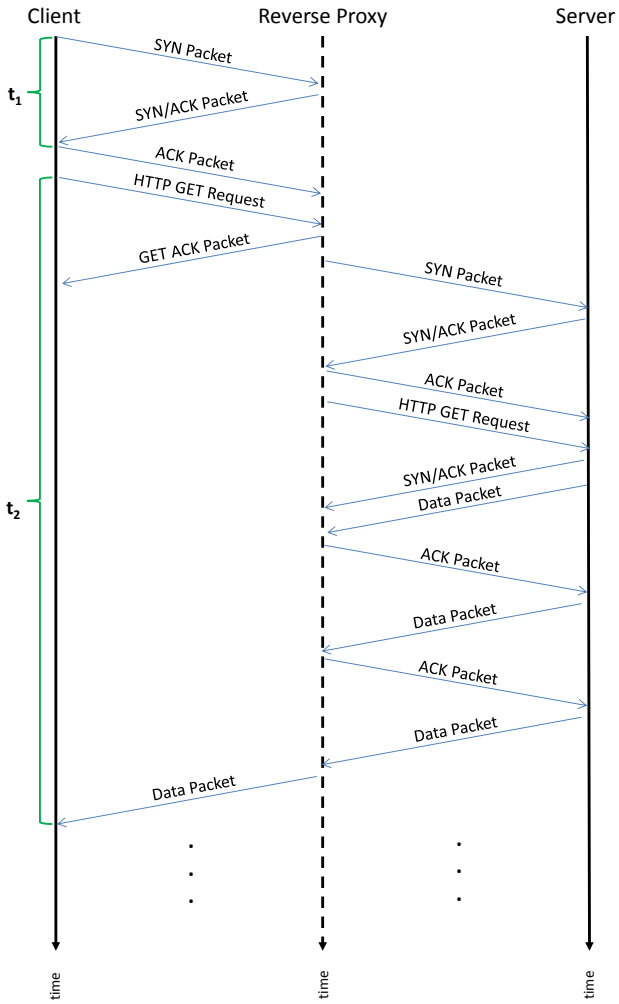


Figure 3.6: Reverse Proxy Connection RTTs

servers to which they were connecting when compared to Apache. Regardless of the browser, both servers would establish the connection via a 3WHS, ACK the GET request, then attempt to connect on the back end to the website. The website would receive the reverse proxy GET request, and occasionally send a 302 message redirecting the client to the same server but a different resource, such as a PHP webpage. The redirection would bypass the nginX and/or Squid reverse proxy servers. An HTTP 302 message is defined as a temporarily relocated resource message that redirects the client to another URL. If the client wants to return to the same website in the future, it should visit the same website it tried in the beginning [21].

We tested both of the servers against multiple websites. The website seemed to be the de-

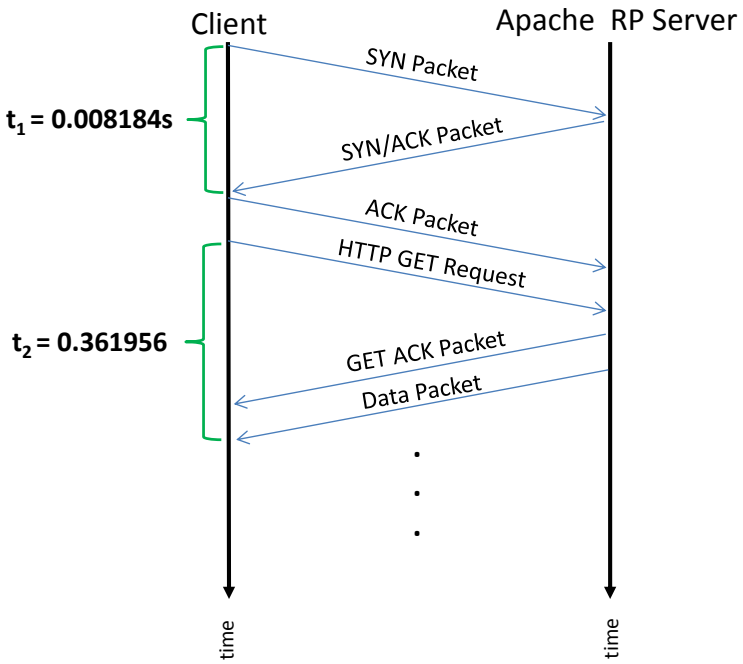


Figure 3.7: Reverse Proxy Timing Diagram from the Client's Perspective using Apache

termining factor as to whether or not a 302 message would be returned through the reverse proxy. For example, Amazon would not submit 302 messages while the United States Naval Academy's website would. The 302 message would get sent to the client via the reverse proxy, leading to a connection termination between the client and the reverse proxy. The client would then make a second connection to the newly specified URL provided within the 302 message (Figure 3.8). Regardless of the response from the back end server, the time required for t_2 is noticably larger than t_1. This characteristic is essential in identifying reverse proxies, which we discuss in the next section.

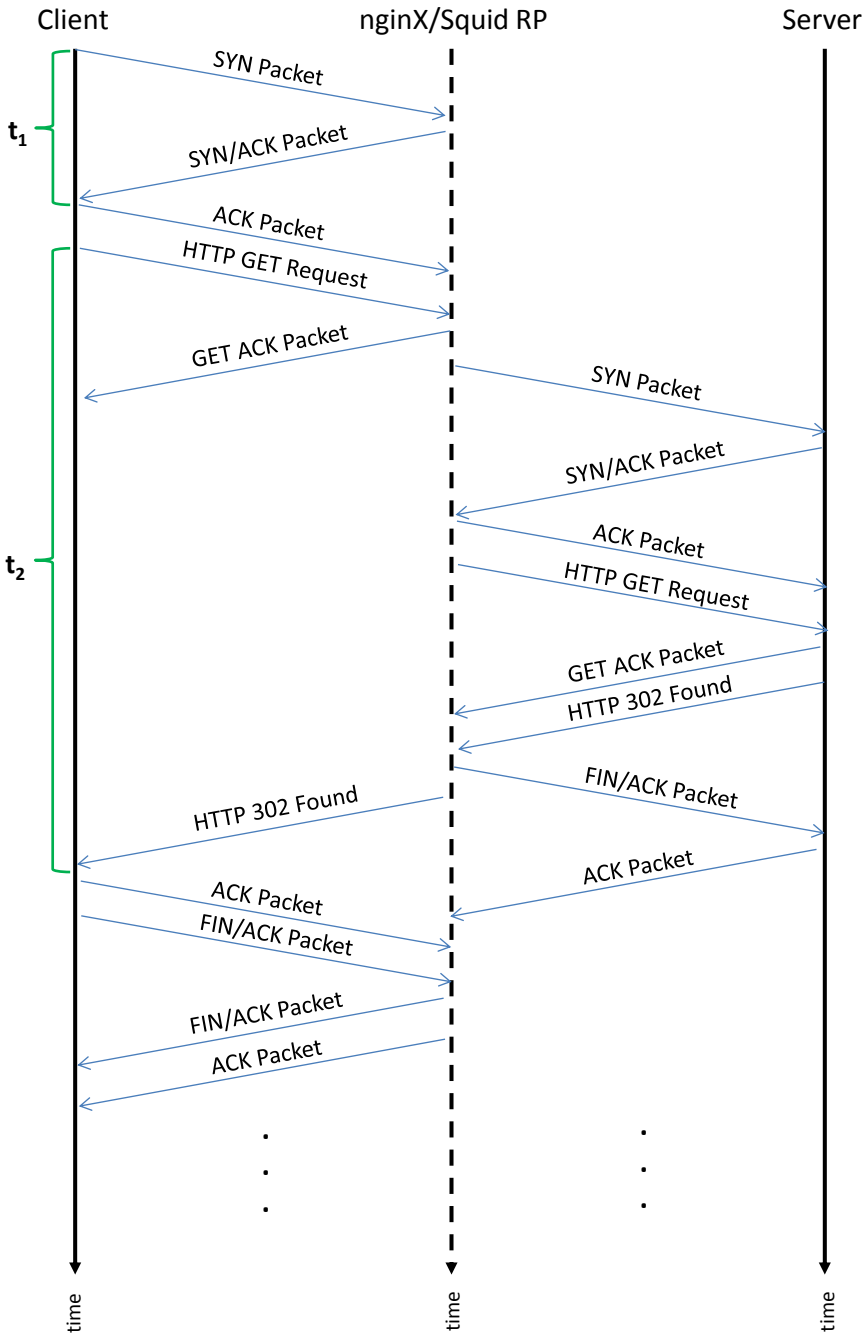


Figure 3.8: Reverse Proxy Timing Diagram from the Client's Perspective using nginX and Squid

3.4 Algorithmic Approach

Manual inspection of several flows revealed that the 3WHS is within a few standard deviations when compared to direct connections as well as reverse proxies. We also see similar packet flow between the same two types of connections. The main difference between a direct connection and a reverse proxy connection is that the time delta between the HTTP GET request and the first data packet from the server is significantly greater for a reverse proxy. RTTs are typically calculated using Karn's algorithm [35], which only takes into consideration the order that packets are sent and received. Specifically, "When an acknowledgement arrives for a packet that has been sent more than once (i.e., retransmitted at least once), ignore any round-trip measurement based on this packet, thus avoiding the retransmission ambiguity problem. In addition, the backed-off RTO for this packet is kept for the next packet. Only when it (or a succeeding packet) is acknowledged without an intervening retransmission will the RTO be recalculated from SRTT" [35]. RFCs 1185 [43] and 1072 [44] cover the current approaches to handling TCP ordering given the high bandwidth connections of present times.The challenge this presents is that our GET RTT will be approximately the same as the 3WHS RTT or smaller regardless of the TCP connection type. The HTTP GET request ACK includes no data, so the reverse proxy has no requirement to connect to the back end web server.

3.4.1 RTT Analysis Tools

Most TCP flow tools that provide RTT analysis use Karn's algorithm [35] to identify RTTs. We initally expected that one of these tools would be able to automate RTT collection for us. We used TCPTrace [45] in an initial experiment to build a database of RTT measurements across thousands of flows. TCPTrace provides a lot of interesting RTT characteristics such as variance, averages, client to server RTTs, and server to client RTTs. One observation we had using TCPTrace is that RTT values are calculated from the server to client. The server to client RTT is calculated as the time elapsed from when an incoming packet is received by the web client to the time the response packet is sent from the web client. This time is solely based on processing delay at the client. The server to client RTT calculations would be most helpful given network traffic capture from a vantage point between the client and server, but our study is focused on client vantage points. RTT times are generally associated as elapsed time for packets in transit due to queueing and propagation delay. Additionally, TCPTrace rounds times to the tenths of milliseconds, which is not a high enough resolution to capture many time stamps based on typical Internet speeds. We also experimented with Tstat [46], which yielded similar RTT results to TCPTrace, and Captcp [47] which was not as helpful as the former two. Each tool we

tested had an RTT estimation process that did not suit our needs. We decided to implement our own approach instead of using a common tool.

Our definition for the 3WHS remains the same as Karn's algorithm. The 3WHS is defined as the time from when a SYN packet is sent from a web client to the time a SYN-ACK packet is received by the web client. This RTT will typically be dominated by propagation delay between the web client and the web server except in some cases where high server load results in large queueing and/or processing delays. We discard Karn's RTT definition for GET RTT calculation in order to include the first data packet from the server on the wire. Our definition for GET RTTs is the time when a GET request is sent from the web client to the time that the first packet that includes data is received (See Figures 3.5 and 3.6). This definition disregards the GET request ACK packet in order to include the time delay for data transfer. The data packet does not have to have the correct sequence number because we only care about receiving a data packet. Data packets require a connection to the back end server for a reverse proxy so out of order data packets are sufficient.

The GET RTT results for direct connections are relatively equivalent to their corresponding 3WHS RTTs. Non-production reverse proxies, such as those used by smaller websites and likely malicious websites, must establish a new connection to the backend server and retrieve some number of data packets before data can be transferred to the web client. It is also important to recognize that many webpages contain embedded links to data such as images, advertisements, tables, and so forth within the HTML code. Embedded links may cause multiple GET requests to occur within a single TCP flow. If multiple GET requests occur within a TCP flow, we apply the same GET RTT definition and count the GET RTT result as additional validation for our timing analysis.

Similar to our approach for test cases, we wanted to use a bottom-up approach to identifying RTTs. We developed a method to obtain the 3WHS and GET RTTs based on packet characteristics in a single flow pcap file so that we could do manual inspection on our results. Once we were able to verify our method worked, we adapted the method to handle multiple flows within a single pcap file.

3.4.2 Single Flow Analysis

In order to identify the 3WHS RTT and the HTTP GET request RTTs, we present pseudocode to step through a pcap file of our own generated network traffic and identify the packet

features that give us the timestamps we need. The first example is shown in Algorithm 1, where we step through a single TCP flow using states and packet directions between the web client and the web server. Since a single flow may have multiple GET requests, we keep track of each GET in an ordered map of GET requests within the map h.

We open a pcap file p, read a single packet at a time, and store the packet data into buf and the packet time stamp into ts. We use a finite state machine (Figure 3.9) to keep track of the flow's current state. Each complete TCP flow establishes a connection, transfers some data, and closes the connection. We use this knowledge to establish states. We instantiate a new flow by setting the state to $START$. First, we read a packet from the pcap file p. Since we are looking at HTTP flows we use a destination port of 80, the established port for HTTP traffic [48], to determine if a packet is inbound or outbound from the web client. If the packet is outbound and the flow is in the $START$ state, we save the time stamp for the first part of the $3WHS\_RTT$ and move in our state diagram to SYN_SENT. We read in a packet, looking for the SYN and ACK flags to be set in the TCP header. If we meet these prerequisites, we set the flow state to SYN_ACK, subtract the first time stamp from the current timestamp ts to calculate the $3WHS\_RTT$, and save the $3WHS\_RTT$ into h, using the $clientPort$ as the key in h. We use $clientPort$ as the key because (1) ports are rarely repeated in close proximity given that there are tens of thousands of allowable port numbers per client and (2) we will limit the number of TCP connections per pcap file p to less than the maximum allowable number of TCP ports.

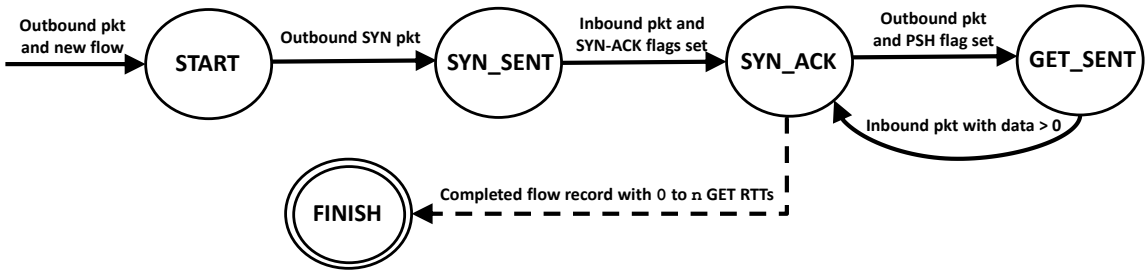


Figure 3.9: FSM for parsing TCP flows with HTTP

The flow has been saved into h at this point. Some flows may not contain any GET requests because of failed connections, time outs, or legitimate TCP connection closed procedures with FIN-ACK packet exchange. The state may finish at this point in the state diagram. However, if we read an outbound packet with the PSH flag set we set the flow state to GET_SENT. The PSH flag is used to to notify TCP to send data immediately from the server side and to push

data to the receiving application immediately from the client side [42]. In this case, the web client used a TCP packet with the PSH flag set in order to send the GET request. We then save timestamp ts as the first marker for a GET_RTT.

If we are in the state GET_SENT and we read an inbound packet that has no data, we identify this as the GET ACK packet. This packet would typically resemble the completion of an RTT, as we mentioned earlier in this chapter. We disregard this packet because it contains 0 bytes of data, and therefore has not yet communicated with the data server if it is a reverse proxy. We read in another packet and check for some amount of data greater than 0 bytes. If we find an inbound packet that contains data, we calculate the GET_RTT by subtracting timestamp ts from our saved timestamp, add the GET_RTT to an array of GET RTTs in our map h using $clientPort$ again as the key, and move our flow state back to SYN_ACK. Moving our flow state back to SYN_ACK allows us to test for more GET RTTs. If we identify more GET RTTs within the single flow in p, they are appended to h using the $clientPort$. If we read through p do not see any more GET RTTs, the flow is already saved in h and we move to the finished state. We may see anywhere from 0 to n GET RTTs for a single flow. This completes a single flow analysis.

We know that multiple flows will be in a single pcap file p because of how we planned to collect our data. HTTP GET requests are likely to have one or more redirection, as discussed by Nolan et al [12]. A redirected connection will initiate a new TCP connection, leading to multiple flows in what was formerly considered a single flow. We do not want to follow redirections to website addresses outside of our targeted website lists for our current research, but may investigate this further in future work. Furthermore, embedded links may not be stored on the same server as the initial HTML which will cause even more TCP connections to be established. We recognize that we must adapt our algorithm to handle multiple flows with interleaving packets.

3.4.3 Multi-flow Analysis

We will certainly have more than one TCP flow per pcap file because of our data collection methodology. We account for this inefficiency by adapting Algorithm 1 to handle multiple flows with interleaving packets in a pcap file. We encompass all of the pseudocode in Algorithm 2, adding map g to handle $flow\ State$ for each flow, filtered by $clientPort$, and adding map i for tracking $flow\ timestamps$. We continue to identify packets as inbound or outbound based on port 80. If we read a packet as outbound, we also check if the $clientPort$ flow exists in g. If it does not exist, we instantiate a new flow in g using $clientPort$ as the key, setting that flow's state to $START$. This allows multiple flows to be tracked based on their individual states.

We modify our state transitions to look at flow state within g using $clientPort$ to keep track. Our FSM remains the same as it did in Algorithm 1 except that we store time stamps in i using $clientPort$ as the key. The i map allows us to track the most recent time stamp for each flow based on $clientPort$. We update i with the packet timestamp ts when we identify a stored flow in our FSM. The rest of the algorithm is carried forward in the same fashion, so we omit the remainder for brevity.

We implemented Algorithm 2 into a program and ran it against several pcap files that we created by generating our own network traffic to various websites. We were able to verify the results by manually inspecting the tested pcap files using Wireshark [41]. We used a calculator and the timestamps from Wireshark to calculate RTT values using the same approach as described in Algorithm 2 for our manual inspection and verification. After numerous test cases were examined of varying size from single flow pcaps to multi-flow pcaps, we began probing lists of websites and collecting network traffic to analyze our methodology.

3.5 Data Collection

The data we wanted to collect needed to be close to an equal distribution of popular websites, known malicious websites, and known reverse proxy servers for comparitive analysis. Furthermore, we wanted to have a variety of vantage points in order to observe whether or not our results were persistent globally. We built a simple script that saves the network traffic generated from each vantage point which we are able to analyze later.

3.5.1 Vantage Points and Network Traffic Generation

We used 10 PlanetLab [49] vantage points to collect our network traffic. These nodes were located in Germany, the United Kingdom, Thailand, China, Japan, and five locations within the United States (Albany, GA, St. Louis, MO, Berkeley, CA, Providence, RI, and Madison, WI). We used nodes that were globally distributed to see if the timing results showed consistent differentiation between the 3WHS and GET RTTs regardless of location. Instead of using domain names for our HTTP requests, which could be influenced by DNS lookups from each vantage point, we used bulk domain name service lookup software to identify domain IP addresses first then execute our network traffic based off of those IP addresses. DNS lookups return different results based on location. We kept the location consistent by resolving IP addresses from Monterey, CA.

3.5.2 Alexa Website Lists

We want to look for differences in timing results across several layers of popular websites. We use Alexa [50] to pick these websites. Alexa is a service provided by Amazon [51] that calculates the top one million most frequently visited websites around the world. We chose to analyze the top 80 websites from Alexa which did not resolve to the same IP address. These websites are generally large companies, and employ IT personnel to manage their servers. This is important because though they may employ reverse proxy servers, the response times within their intranets or server farms will likely be incredibly fast. As a result, it will be more difficult to identify reverse proxies for these websites when compared to the other selected website lists.

Secondly, we used a random number generator to draw Alexa ranks from the uniform distribution of Alexa's 5k to 10k websites for our semi-popular website list. We expect that these websites may show lower performance with response times. The types of websites we see in this range include universities, smaller companies, and foreign websites that may be limited in scope to users within their country of origin. Lastly, we use the same random number generator to collect 100 numbers between 500,000 and 1,000,000 to get some websites in the top 0.5% of all websites. There are roughly 700 million websites on the Internet, but only around 200 million websites are active. We expect that this bottom tier of websites may show similar results to our malicious website list. Several websites look like the type of URLs one would see in a spam list, such as `drugadmin.com`, `rxglobalmeds.com`, or `cheaprolexsale.biz`. Furthermore, some websites may be used for botnet command and control servers using HTTP as the communication channel, possibly even reverse proxies.

3.5.3 Malicious Website List

We needed to have a collection of known malicious websites. We searched the Internet for black lists or malicious domain lists, and found an IP address listing of malicious domains at `http://www.malwaredomainlist.com`. This list included over one thousand different malicious domains, and was updated regularly. We chose to run our network traffic generator against 460 of these domains because of time constraints on collecting the data. We chose to limit the number of retries per HTTP request to 1, and to limit the timeout value for each request to 30 seconds. We wnated to limit the amount required for each timeout without cancelling longer delayed connections. Websites that no longer existed would create large delays for our collection script, which is where the bulk of our time constraints originated since we collected sequentially. Over one hundred of the domains were no longer active, so no connections were

established. An estimated 300 malicious domains allowed connections for us to examine. The malware domain list keeps an up-to-date record of each website by domain and IP address with a description of why it is classified as malicious. Some of the examples include known web browser exploits, trojan horses, or ZIP files that have embedded exploits. If we are able to identify any of these malicious websites as reverse proxies, the background information for the targeted website may be helpful in grouping similarities in reverse proxy servers.

3.5.4 Known Reverse Proxy List

We used a reverse proxy server that we controlled locally in conjunction with 10 known reverse proxy servers. Our personally controlled reverse proxy server was built using Apache [19], with the back end target of GoDaddy [40] that was located across the country from our reverse proxy location. A typical reverse proxy will be in relatively close proximity to its internal data servers. We wanted to have a large delay to help with identification thresholds which we discuss in the next chapter. The additional 10 reverse proxy servers were gathered using Torrent proxy servers based in Europe.

Algorithm 1 Single Flow Collection with Round Trip Times Using State and Packet Direction

```
1: Open pcap file p;
2: Create map h;
3: Initialize State to START;
4: for each packet in p, read into buf along with timestamp ts do
5:    if (packet is not IP) then
6:       continue;
7:    Read the packet's Ethernet and IP headers;
8:    if (protocol number is not TCP) then
9:       continue;
10:   Read TCP header;
11:   if (dstPort is equal to 80) then
12:      Direction is Outbound;
13:   else
14:      Direction is Inbound;
15:   if (Direction is Outbound and State is START) then
16:      if (SYN flag is not set) then
17:         continue;
18:      State is set to SYN_SENT;
19:      firstTS = ts;
20:      continue;
21:   if (Direction is Inbound and State is SYN_SENT) then
22:      if (SYN and ACK flags are not set) then
23:         Packets are out of order, flow will not process;
24:      State is set to SYN_ACK;
25:      3WHS_RTT = ts - firstTS;
26:      h.add(clientPort, [clientIP, serverIP, 3WHS_RTT, 0]);
27:      continue;
28:   if (Direction is Outbound and State is SYN_ACK and PSH flag is set) then
29:      State is set to GET_SENT;
30:      firstTS = ts;
31:      continue;
32:   if (Direction is Inbound, State is GET_SENT, and data > 0) then
33:      State is set to SYN_ACK;
34:      GET_RTT = ts - firstTS;
35:      if (h.find(clientIP) returns true) then
36:         h.update(clientPort, [clientIP, serverIP, 3WHS_RTT, GET_RTT]);
37:      else
38:         The flow does not exist ... error;
39:         continue;
```

Algorithm 2 Multiple Flow Collection with RTTs Using State and Packet Direction

```
1: Open pcap file p;
2: Create map h for tracking flow RTTs;
3: Create map g for tracking flow State;
4: Create map i for tracking flow timestamps;
5: for each packet in p, read into buf along with timestamp ts do
6:    if (packet is not IP) then
7:       continue;
8:    Read the packet's Ethernet and IP headers;
9:    if (protocol number is not TCP) then
10:      continue;
11:   Read TCP header;
12:   if (dstPort is equal to 80) then
13:      Direction is Outbound;
14:      if (g.find(clientPort) returns false) then
15:         g[clientPort] = START;
16:   else
17:      Direction is Inbound;
18:   if (Direction is Outbound and g[clientPort] = START) then
19:      if (SYN flag is not set) then
20:         continue;
21:      g[clientPort] = SYN_SENT;
22:      i[clientPort] = ts;
23:      continue;
24:   if (Direction is Inbound and g[clientPort] = SYN_SENT) then
25:      if (SYN and ACK flags are not set) then
26:         Packets are out of order, flow will not process;
27:      g[clientIP] = SYN_ACK;
28:      3WHS_RTT = ts - i[clientPort];
29:      h.add(clientPort, [clientIP,serverIP,3WHS_RTT,0]);
30:      continue;
31:   . . .
32:   [continue similarly to Algorithm 1];
33:   . . .
```

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
RESULTS AND ANALYSIS

4.1 Overview

We collected 11 gigabytes of network traffic captures that were grouped into five sets. Three of the sets were selected from Alexa's top one million websites. We selected 80 websites from the top 100, 100 websites from the 5,000 to 10,000 range, and 100 websites from the 500,000 to 1 million range. We also had a group of known malicious websites and a list of reverse proxy websites, including one that we controlled. The following sections will describe how we collected the data, parsed all of the network traffic data into readable and analyzable formats, and then present the results and analysis to verify our hypotheses.

4.2 Data Collection, Parsing, and Organization

Once we finished collecting network traffic data, we knew we needed to have a concise approach to storing flow results. The obvious requirements for storing flow data included the 3WHS and GET RTTs along with a minimum of one data point to differentiate between flows. The differentiating value follows from our algorithm, using the client port to distinguish between flows. When we write hundreds or thousands of flow results into a single data file, we need more unique flow factors for analysis than the three aforementioned features. We added the source and destination IP addresses so we could see from which PlanetLab node the traffic originated and the targeted website for each flow. In the cases where a flow had more than a single GET RTT because the same TCP connection was used to retrieve multiple objects, we kept a counter for the number of GET RTTs so that we could average the GET RTT time for a single flow. We used this average GET RTT for calculating flow RTT ratios, which we will discuss further in this section.

4.2.1 Data Collection

We used a bash script to collect our network traffic data from each PlanetLab Node. The script would read in IP addresses from a file passed in as an argument. The IP addresses were targeted by our script one at a time, sequentially executing the linux wget tool 250 times, then saving the network traffic for a single website per node for analysis. We used tcpdump to collect the network traffic. Each pcap file contained the network traffic from a single PlanetLab

node to a single website. We could then take each pcap file and analyze the results as an individual case. Furthermore, we could store the pcap files into groups depending on the filter we wanted to use. The types of filters that we may want to use to organize our data could be by PlanetLab node, website group, or website. We chose to break our data into website groups and PlanetLab Nodes. Overall, this resulted in 950,000 network traffic captures for the Alexa websites, 750,000 captures for the malicious websites, and 27,500 captures for known reverse proxies between June and September 2013.

4.2.2 Reading Traffic Captures into Table Entries

We implemented Algorithm 2 into a C++ program that analyzes a single pcap file and writes flow data to a file using comma-separated values (CSV) format. Each line in the CSV file represents a single flow. An example of this can be seen in Table 4.1, where the total row count equals the number of flows for that data file. We use a bash script to automate our C++ program across all pcap files from a PlanetLab Node within a single file directory. In this way, we can create a single data table for a pcap file that contains all flow results within that pcap. The way that we collected network traffic was between a single host and destination for each pcap file, initiating 250 web requests to each destination.

Table 4.1: Data Storage for Network Traffic Capture Results - Flow per Row

Client Port	Client IP Address	Server IP Address	3WHS RTT	# of GET RTTs	GET RTT[1]
44034	169.226.40.2	74.113.233.48	0.015901	1	0.016551
44037	169.226.40.2	74.113.233.48	0.018353	1	0.016233
43783	169.226.40.2	74.113.233.48	0.015374	1	0.016882
...	...	...	...	...	...

We only captured a maximum of 249 flows per pcap file because our network capturing program, tcpdump, took more time to begin collecting packets than we allowed before starting our web requests. The single missed flow is negligible. Some target websites showed connection challenges. We use our malicious website list and the Albany PlanetLab node as an example (Table 4.2). Only 221 flows were collected out of the 249 expected because a connection could not be established. Additionally, we found that data retrieval occasionally fails even though the initial connection is successful. The malicious website set showed a higher percentage of failed connections as well as failed GET requests when compared to the Alexa website lists. All sets typically had 1 to 5 failed GET requests per 250 requests. Manual inspection shows that the failed GET request flows had packet transfers as depicted in Figure 4.1.

There were several thousand CSV data files once we finished running our script and C++

Table 4.2: Albany PlanetLab Node Traffic to Malicious Website: 178.73.224.97

Client Port	Client IP Address	Server IP Address	3WHS RTT	# of GET RTTs	GET RTT[1]
58688	169.226.40.2	178.73.224.97	0.105189	1	0.304336
58944	169.226.40.2	178.73.224.97	0.105583	0	-
54337	169.226.40.2	178.73.224.97	0.105192	1	0.303039

program against all of our pcap files. These files provided a raw look at the packet timing between a client and server which we could aggregate to get finer grain detail for analysis. In particular, we wanted to be able to look at the cumulative distribution of 3WHS to GET RTTs from a client to a server as a single data entry. Our solution was to use a ratio of the GET RTT to the 3WHS RTT.

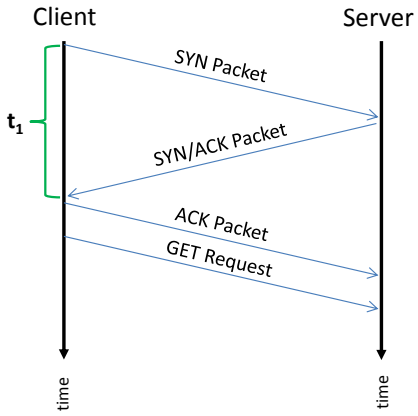


Figure 4.1: Packet Timing Diagram for Failed GET Requests

4.2.3 Building Flow Sets

We obtained the average GET RTT by using the number of GET RTTs column combined with the GET RTT timing values in the CSV data file. We divide the average GET RTT by the 3WHS RTT to have a single flow RTT ratio. This ratio should be roughly 1:1 for a direct connection as we discussed in Chapter 3. The expected ratio for a reverse proxy server would be greater than 1:1, but we had to analyze the known reverse proxy results in order to establish a threshold. Tables 4.3 and 4.4 show three separate entries in our summarized data file. A single entry has the identifying flow information with the client and server IP addresses as well as the server domain name, if any, in the first three columns. If the server does not have a domain name, as is the case for most of our malicious websites, it is labed as an 'undefined domain'.

The fourth and fifth columns track the total number of flows identified and the fraction of

total flows that contained GET RTTs. This requires some explanation. We knew that 250 web requests were initiated for each pcap file. If we identified only 240 flows that established connections, we would put 240 in the fourth column. We would take the total number of flows that had GET RTTs and divide that by the number in column four to get the fraction in column five. This gave us an idea of how many connections were actually established between the client and server, as well as the those connections that were able to successfully retrieve data with the server.

Table 4.3: Data Storage for Cumulative Distribution of Flow Ratios for a Single Vantage point: Columns 1-5

Client IP Address	**Server IP Address**	**Server Domain Name**	**Total Number of Collected Flows**	**Fraction of Total Requests with a GET RTT**
169.226.40.2	101.1.18.24	ued123.cn	249	1.0
169.226.40.2	108.162.198.182	professionalcrew.net	249	1.0
169.226.40.2	109.69.186.62	monsieur-meuble.com	249	1.0

Table 4.4: Data Storage for Cumulative Distribution of Flow Ratios for a Single Vantage point: Columns 6-11

Average Ratio	**Maximum Ratio**	**10th Percentile**	**25th Percentile**	**75th Percentile**	**90th Percentile**
1.00850709519	2.0050137043	1.0565533638	1.02593505383	1.00303089619	0.988574624062
1.11531659039	2.88584566116	1.19612193108	1.11458528042	1.06560087204	1.03861892223
1.08726804611	13.0100908279	1.0081192255	1.00470817089	1.00184178352	0.999478459358

The sixth column is calculated by averaging all of the RTT ratios for the pcap file. The maximum ratio is identified by sorting all of the ratios in ascending order in an array and taking the value in the largest array index. The eighth through eleventh columns are important for our flow analysis in the next section. We take our sorted ratios array and identify the index in the array that correlates with the percentile we want. The percentile values allow us to analyze the ratio spread across all requests to a given website from a single client.

Our flow analysis is broken into two sections. The first section will cover our findings looking at the website sets holistically. We will show how the results match up with our hypotheses in Chapter 3 regarding set correlation or divergence. The second section will be the analysis for our reverse proxy threshold and the statistics for websites that match this threshold. Lastly, we will cover several detailed case studies on websites that show interesting signatures.

4.3 Website Group Results

In this section we present the results and analysis for each website group holistically. We start with the Alexa website groups, leading off with Alexa's top 100 websites. The final group we analyze is the malicious websites group. The results for this section show us a high-level view for timing ratio results across all website groups, and allow us to identify the percentages of websites within each group that may be identifiable as a reverse proxy. The percentile results for each website group provide a filtered estimation for timing ratios that we can use to compare with our known reverse proxy results in the next section. Overall, the vantage points were consistent for every group which carried through to the case studies. A single outlier existed with Singapore's PlanetLab node results against `feedblitz.com` which we covered in a case study.

4.3.1 Alexa's Top 100 Websites

The Alexa Top 100 websites include major search engines (i.e., Google, Yahoo, Baidu), online commercial companies (i.e., Amazon, eBay, Apple), and social networking or blogging sites (i.e., Blogspot, Facebook, Twitter). Over 20 of the top 100 websites resolve to the same IP address, so we only use 79 of the websites from the top 100. To be consistent with our referencing, we will continue to refer to this group of websites as the Alexa Top 100.

The top 100 Alexa Websites data contained a total of 191,754 flows. 183,892 flows had GET RTTs that we used to calculate RTT ratios, equaling 95.9% of what we initiated as web requests. Figure 4.2 depicts the 10th, 25th, 50th, 75th, 90th, and 100th percentiles as CDFs. Referring to Table 4.5, we can see the first value for each percentile CDF. The lowest curve represents the 10th percentile whereas the highest curve represents the 100th percentile. The lower 90% of these flows had a ratio under 2.003.

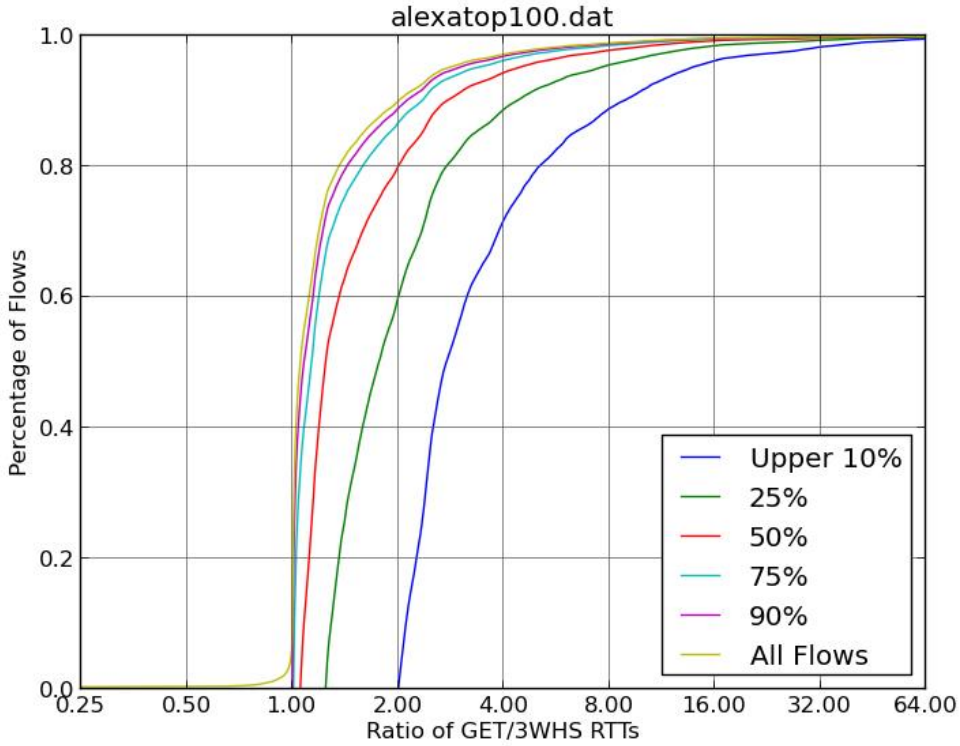


Figure 4.2: Alexa Top 100 Websites Cumulative Distribution of Ratios

Table 4.5: Percentile Values for Alexa's Top 100 Websites

Average Ratio	Maximum Ratio	10th Percentile	25th Percentile	50th Percentile	75th Percentile	90th Percentile
1.53292146618	1307.11242676	2.00307941437	1.2427624464	1.05298805237	1.00675189495	0.999756515026

We appended the individual node results into a single data file formatted as Tables 4.3 and 4.4 so we could sort the data for easier observation. We refer to a single entry in this resulting file as a *flow set*. A *flow set* consists of all flows (up to 250 flows) collected from one PlanetLab vantage point and one destination website. We could see 788 distinct flow sets in the our results. We identify three websites that establish zero connections across all of the PlanetLab nodes. Each number shown in Table 4.6 shows the fraction of flow sets, out of the total 788, with at least a certain percentage of the flows having their GET-3WHS ratios greater than a candidate threshold value.

Generally speaking, a higher value of T corresponds to a higher confidence than the websites in the marked flow sets exhibit the use of reverse proxies from one or more vantage points. The same is true with respect to the value of X. We know that higher values of X refer to the higher ratios within flow sets so the percentage of flow sets out of 788 will be higher with lower values of X. For instance, when X is equal to 10%, we know we are looking at all flow sets' largest 10% of ratios. Conversely, if we look at high values of T and high values of X, we see smaller percentages of flow sets that meet the criteria because we are including the smaller ratios from all flow sets. We omit the lowest 10% of flows to counter possible server caching affects on our results. Though an X value of 100 that still meets a high T threshold is the strongest sign of a reverse proxy, the lower ratios may force a website to be discarded which is undesirable.

Table 4.6: Percent of Top 100 Alexa Flow Sets within Ratio Thresholds

# Flow Sets with at least X% flows having ratio > T	**Threshold T = 1.25**	**Threshold T = 1.50**	**Threshold T = 1.75**	**Threshold T = 2.00**	**Threshold T = 2.25**
X = 10	245 / 788 = 31%	158 / 788 = 20%	124 / 788 = 16%	97 / 788 = 12%	87 / 788 = 11%
X = 25	185 / 788 = 23%	126 / 788 = 16%	97 / 788 = 12%	80 / 788 = 10%	61 / 788 = 8%
X = 75	144 / 788 = 18%	97 / 788 = 12%	73 / 788 = 9%	54 / 788 = 7%	41 / 788 = 5%
X = 90	135 / 788 = 17%	91 / 788 = 12%	67 / 788 = 9%	42 / 788 = 5%	32 / 788 = 4%

A single table entry (Table 4.6) may be multiple counts of a single website but from a variety of vantage points. We can pair down the numbers in Table 4.6 by website to get the total number of websites given an RTT ratio threshold to be more exact. One particular website had the highest ratios consistently across all percentiles. `qq.com` had average RTT ratios of 7.8 and higher across all PlanetLab nodes. If we factor in the prevalence of `qq.com` with Table 4.6, we can see that a single website can make up a large percentage of the table representation. `Apple.com` also had interesting results across all nodes. Upon further inspection, we used a legacy server IP address for Apple that redirected all traffic to a separate location. The fingerprint of this website and other similar websites in the Alexa Top 100 group will be compared to that of reverse proxy timing in the next section. Additionally, we will examine several websites such as `qq.com` in more detail in the case study examples.

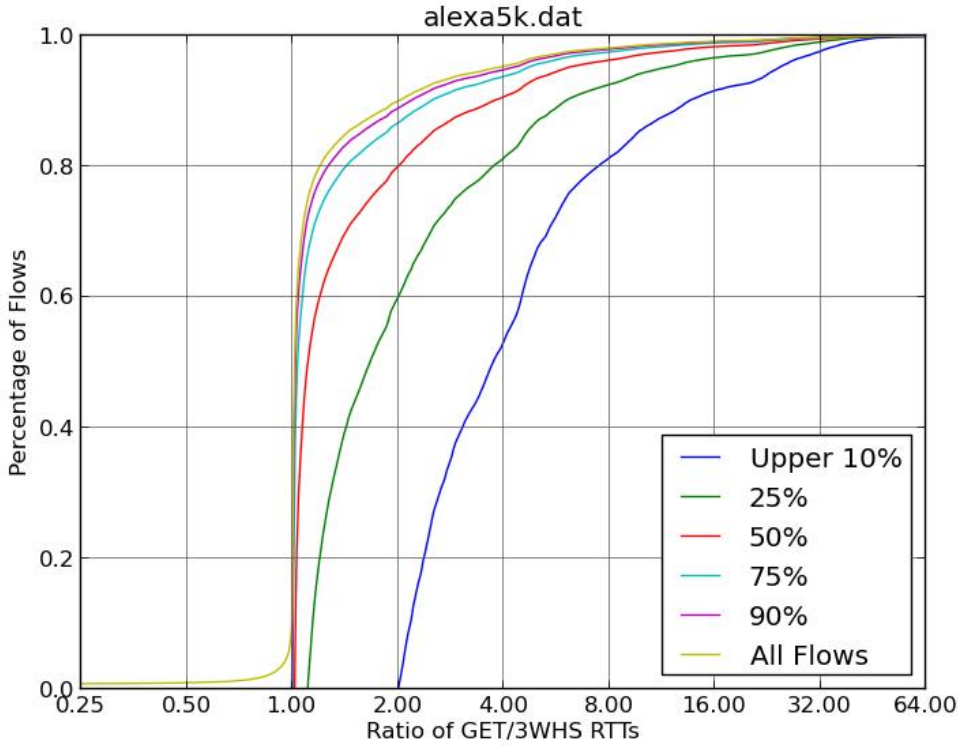


Figure 4.3: Alexa 5,000 to 10,000 Websites Cumulative Distribution of Ratios

4.3.2 Alexa's Top 5,000 to 10,000 Websites

The types of websites we see in the Alexa listing between ranks 5,000 to 10,000 include educational institutions (i.e., Brigham Young University, Duke University, `Education.com`), news websites (i.e., Sky News Arabia, Kemalist Gazete, Detroit News), and entertainment industry sites (i.e., HBO GO, LeoVegas Online Casino, FreeRide Games). The variety of website themes is more diverse than the few types we noticed in the Alexa top 100. We can see a difference in RTT ratio results by comparing Figures 4.2 and 4.3. The larger 10% of RTT ratios for Alexa's 5k-10k group show a larger range of values (Figure 4.3) than the websites in Alexa's top 100, which exhibit consistently low ratios except for the last 5% of total flows (Figure 4.2). More specifically, there are 127 flow sets in the Alexa 5k-10k group that have ratios of 2.0 or above whereas Alexa top 100 only has 97.

Alexa's top 5,000 to 10,000 websites data set had 249,918 flows. 246,917 flows had GET RTTs that we used to calculate RTT ratios, equaling 98.8% of what we initiated as web requests. Figure 4.3 depicts the 10th, 25th, 50th, 75th, 90th, and 100th percentiles as CDFs. Referring to

Table 4.7, we can see the first value for each percentile. The lowest curve represents the largest 10% of RTT ratios whereas the highest curve represents the full collection of flows and RTT ratios. The lower 90% of these flows had a ratio under 2.001.

Table 4.7: Percentile Values for Alexa's 5,000 to 10,000 Websites

Average Ratio	Maximum Ratio	10th Percentile	25th Percentile	50th Percentile	75th Percentile	90th Percentile
1.607620017	285.401397705	2.00146484375	1.1055123806	1.0147652626	1.0021417141	0.995370566845

Similar to our data sorting step in the previous subsection, we count 996 flow sets for our ten PlanetLab Nodes and the Alexa Websites between 5,000 and 10,000. Table 4.7 shows us that 54 flow sets had a ratio higher than 2.25 when we look at the largest 90% of the 250 flows in that flow set. Phrased another way this means that across 90% of the flows between a PlanetLab node and a website the RTT ratio was higher than 2.25 for 54 total node to website pairs. The information provided gives us a starting point for identifying websites that may be reverse proxies since the majority of flows in a flow set consistently have high RTT ratios. There are four consistently high websites within the 54 / 996 grouping. We will discuss one of them in the reverse proxy fingerprinting section later in this chapter.

Table 4.8: Percent of Alexa 5k-10k Flow Sets w/in Ratio Thresholds

# Flow Sets with at least X% flows having ratio > T	Threshold T = 1.25	Threshold T = 1.50	Threshold T = 1.75	Threshold T = 2.00	Threshold T = 2.25
X = 10	254 / 996 = 23%	178 / 996 = 18%	155 / 996 = 16%	127 / 996 = 13%	114 / 996 = 11%
X = 25	186 / 996 = 19%	143 / 996 = 14%	120 / 996 = 12%	102 / 996 = 10%	85 / 996 = 9%
X = 75	133 / 996 = 13%	96 / 996 = 10%	83 / 996 = 8%	67 / 996 = 7%	59 / 996 = 6%
X = 90	123 / 996 = 12%	90 / 996 = 9%	75 / 996 = 8%	58 / 996 = 6%	54 / 996 =5%

4.3.3 Alexa's Top 500,000 to 1,000,000 Websites

Alexa's top 500,000 to 1,000,000 websites cover a large variety of themes. 41 of 100 websites in this group were international domain names (i.e., *.fr, *.ru, *.de). Furthermore, many of the remaining websites with a ".com" domain were listed in foreign languages (i.e., seoders-

lerim.com, ilheusinfo.com, eskisehir.com). When we compare Figures 4.3 and 4.4 we can see much less variation than when we compared Alexa's top 100 to Alexa's 5k-10k groups. The only notable difference between Alexa's 500k to 1 million website group and Alexa's 5k-10k group is that the top 10% and 25% of Alexa's 500k to 1 million group have slightly higher ratios in 80% of the flows.

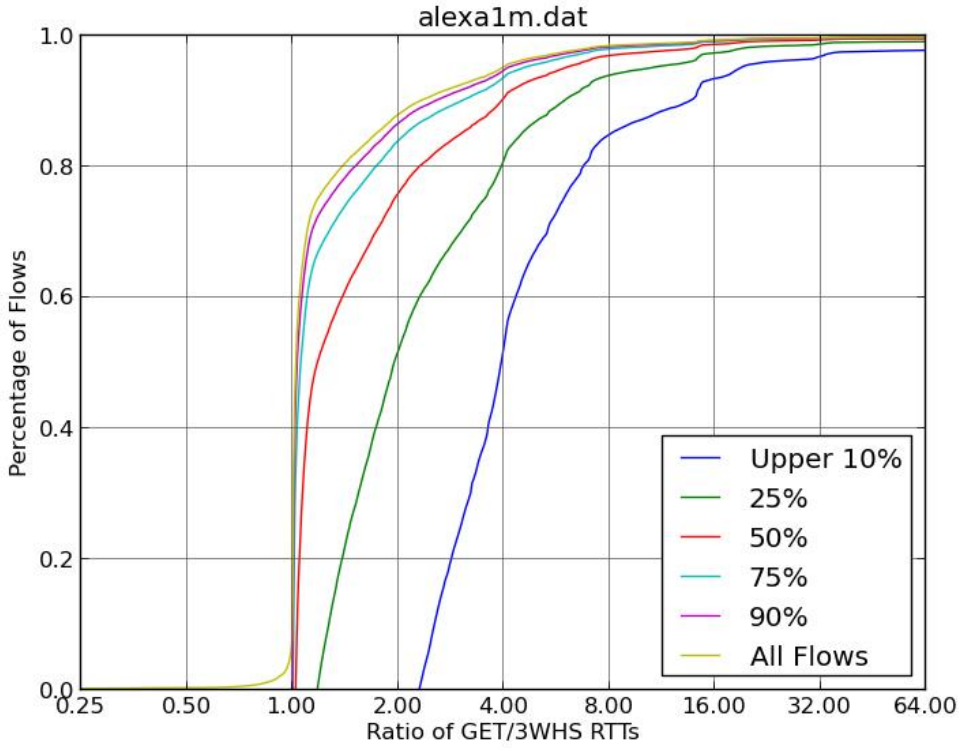


Figure 4.4: Alexa 500,000 to 1,000,000 Websites Cumulative Distribution of Ratios

Alexa's 500,000 to 1,000,000 websites data set contained a total of 238,165 flows. 235,545 flows had GET RTTs that we used tocalculate RTT ratios, equaling 98.9% of what we initiated as web requests. Figure 4.4 depicts the 10th, 25th, 50th, 75th, 90th, and 100th percentiles as CDFs. Referring to Table 4.9, we can see the first value for each percentile.

Table 4.9: Percentile Values for Alexa's 500,000 to 1,000,000 Websites

Average Ratio	Maximum Ratio	10th Percentile	25th Percentile	50th Percentile	75th Percentile	90th Percentile
1.91173319044	733.114562988	2.29619431496	1.17700457573	1.02160072327	1.00476419926	0.999192893505

We counted 959 flow sets for the Alexa 500k to 1m website group. `Eskisehir.com` did not allow any connections via our script, along with 3 other websites in our list. These 4 websites make up for 40 of the 41 lost connections out of 1000. We should have had 1000 given that we used 10 PlanetLab nodes to collect network traffic on 100 websites. `Accuenergy.com`, `oreno.co.jp`, `Aerial7.com`, `Likvidacija-ooo.ru`, and `GetRentalCar.com` dominated the largest ratios in this website group across all 10 nodes, accounting for 40 of the 68 flow sets using an RTT ratio threshold of 2.25. We do a case study on `GetRentalCar.com` in a later section because it had 9 out of 10 flow sets in this highest ratio threshold.

Table 4.10: Percent of Alexa 500k-1m Flow Sets w/in Ratio Thresholds

# Flow Sets with at least X% flows having ratio > T	**Threshold T = 1.25**	**Threshold T = 1.50**	**Threshold T = 1.75**	**Threshold T = 2.00**	**Threshold T = 2.25**
X = 10	263 / 959 = 27%	199 / 959 = 21%	166 / 959 = 17%	139 / 959 = 14%	117 / 959 = 12%
X = 25	214 / 959 = 22%	168 / 959 = 17%	142 / 959 = 15%	111 / 959 = 12%	95 / 959 = 10%
X = 75	186 / 959 = 19%	146 / 959 = 15%	115 / 959 = 12%	93 / 959 = 10%	77 / 959 = 8%
X = 90	170 / 959 = 18%	141 / 959 = 15%	107 / 959 = 11%	85 / 959 = 9%	68 / 959 = 7%

4.3.4 Malicious Websites

The malicious websites that we targeted were based off of a continuously updated website, Malware Domain List [52]. This website provides malicious website entries with date of reported activity, domain name, IP address, description of malicious action, registrant, autonomous system number, and country of origin. The types of malicious activity include a variety of purposes; ransomware, trojans, exploit kits, malicious redirection to compromised websites, java exploits, viruses, and other similar functions. There are over 1000 websites that are tracked as active. We collected network traffic on 300-400 of the websites. The range is variable because of slower connections dependent on the PlanetLab vantage point. We are still collecting data on the remaining malicious websites in the list for future work.

The malicious website data set contained a total of 463,664 flows. 404,315 flows had GET RTTs that we used to calculate RTT ratios, equaling 87.2% of what we initiated as web requests. The large percentage of failed requests may be attributed to websites being identified and taken

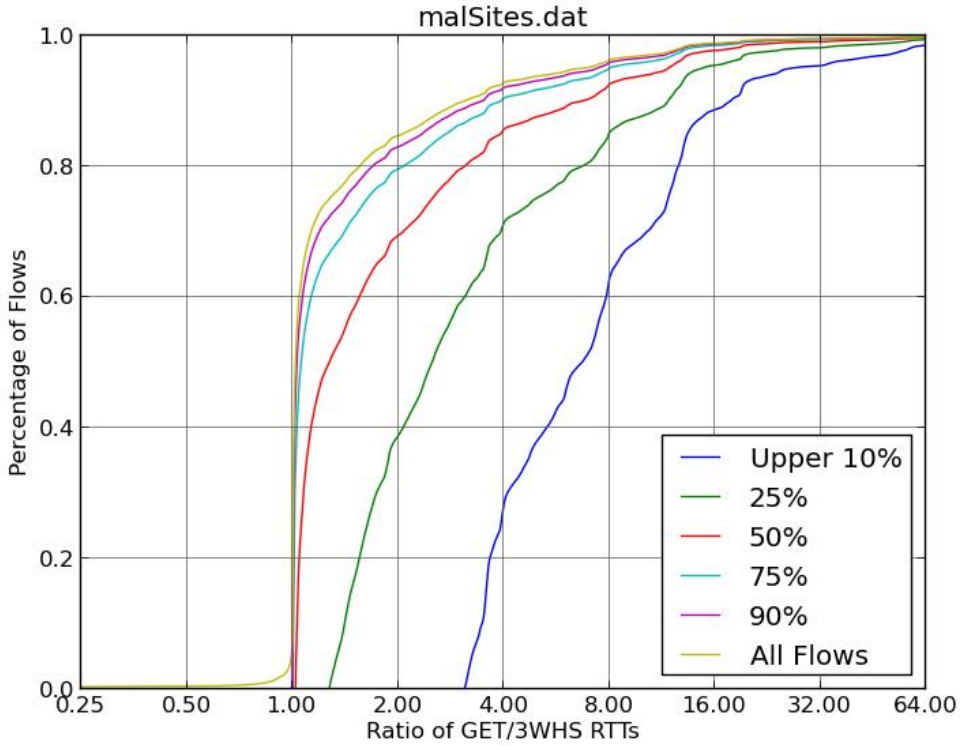


Figure 4.5: Malicious Websites Cumulative Distribution of Ratios

down my domain administrators, or relocated to avoid interruptions to malicious operations. Table 4.11 shows a much higher average ratio compared to the other website groups. Figure 4.5 reflects that roughly 60% of all flows have an RTT ratio near 1. The last 20% of all flows, or those flows with the highest RTT ratios, are the set of websites we will analyze in the case studies in the later section of this chapter.

Table 4.11: Percentile Values for the Malicious Websites

Average Ratio	Maximum Ratio	10th Percentile	25th Percentile	50th Percentile	75th Percentile	90th Percentile
2.08384281878	1130.85083008	3.09227824211	1.27307891846	1.02019119263	1.0042399168	0.999864161015

We counted a total of 1851 flow sets for our malicious website group. The results in Table 4.12 show higher percentages of ratios meeting thresholds when compared to all other website groups (Figures 4.6, 4.8, and 4.10). More specifically, the largest 90% of flow sets using a threshold of 2.25 is 5% higher than the next highest website group (Alexa 500k to 1m). The 218 flow sets within the aforementioned parameters have a few interesting characteristics.

The `149.47.0.0/16` subnet make up 78 of the 218 flow sets. WhoIs lookups for the `149.47.0.0/16` network point to a web hosting company named Precipice [53]. The malicious website locations are geolocated to various positions around the United States (i.e., New York City, NY, San Francisco, CA, Carson City, NV). The Precipice website is a plain website with little functionality. There are several pages that speak about the services and company but there aren't any secure methods to sign up for service or contact the company. The contact page has a simple form that can be filled out and submitted to the company, though the page has no encryption. Overall, the Precipice company looks like a poorly managed web-hosting service that can be used by malicious actors without much scrutiny. Further research on the company background yields phishing reports under the Precipice AS [54] (AS36444).

Table 4.12: Percent of Malicious Site Flow Sets w/in Ratio Thresholds

# Flow Sets with at least X% flows having ratio > T	**Threshold T = 1.25**	**Threshold T = 1.50**	**Threshold T = 1.75**	**Threshold T = 2.00**	**Threshold T = 2.25**
X = 10	531 / 1851 = 29%	427 / 1851 = 23%	359 / 1851 = 19%	314 / 1851 = 17%	291 / 1851 = 16%
X = 25	452 / 1851 = 24%	373 / 1851 = 20%	310 / 1851 = 17%	283 / 1851 = 15%	267 / 1851 = 14%
X = 75	414 / 1851 = 22%	336 / 1851 = 18%	276 / 1851 = 15%	263 / 1851 = 14%	237 / 1851 = 13%
X = 90	401 / 1851 = 22%	318 / 1851 = 17%	269 / 1851 = 15%	247 / 1851 = 13%	218 / 1851 = 12%

The majority of IP addresses within the highest threshold group geolocate to servers in the United States. Other countries within the same group include the United Kingdom, Sweden, and Russia. We will do further analysis on three of the websites within the highest threshold that show consistently high ratios across all percentiles later in this chapter.

4.4 Reverse Proxy Identification

We controlled a single reverse proxy server that was built on Apache [19] and located in Monterey, CA. We set up the internally connected website to be across the country to increase

delays. The increased delay would give us a strong signature for our timing analysis to help with assessing an appropriate threshold. Reverse proxy connections are unlikely to have such large distances to data centers, but matching websites give us higher confidence in the fingerprint. To increase our data set for reverse proxies, we used a website that advertised the free use of "web proxies." These web proxies were provided to execute anonymous downloads through TorrentProxies [55] and its affiliates. Our intuition was that these websites were not reverse proxies, but forwarding proxies to allow anonymized web traffic to subvert illegal activities. Instead of discarding the websites, we added ten of the advertised proxies to our reverse proxy website list and ran our traffic captures. The following section will highlight the results of the reverse proxy tests and present our threshold analysis and comparison to motivate our choices for case studies.

4.4.1 Reverse Proxy Website Group

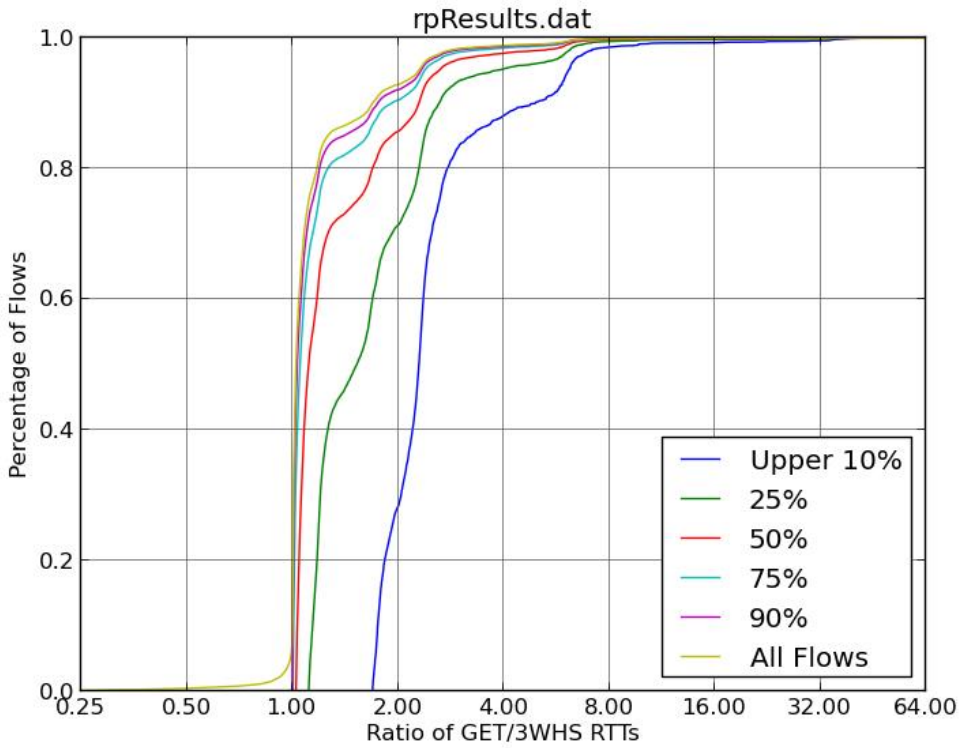


Figure 4.6: Reverse Proxy Websites Cumulative Distribution of Ratios

Once we conducted the initial tests against our reverse proxy lists we observed results unlike we expected. Referring to Figure 4.6, we can see that roughly 80% of all the flows had an

RTT ratio of 1. This would imply that our reverse proxies served as direct connections. This is possible, but we believed it to be unlikely given our base case experiments. Upon further analysis of the actual network traffic for the websites from TorrentProxies [55], we could see that the traffic was being handled as a direct connection. We also tested several of the websites in a common web browser to see what website was rendered. The returned web page was an error from the proxy service provider stating that direct connections to the server were not allowed.

If we use one of the listed proxy addresses in our Internet options, we can anonymously browse the Internet or download illegal content. The latter is the primary purpose for Torrent-Proxies [55]. These proxies were actually forwarding proxies instead of reverse proxies. We had to limit our reverse proxy threshold analysis to the 10 flow sets against our own reverse proxy server.

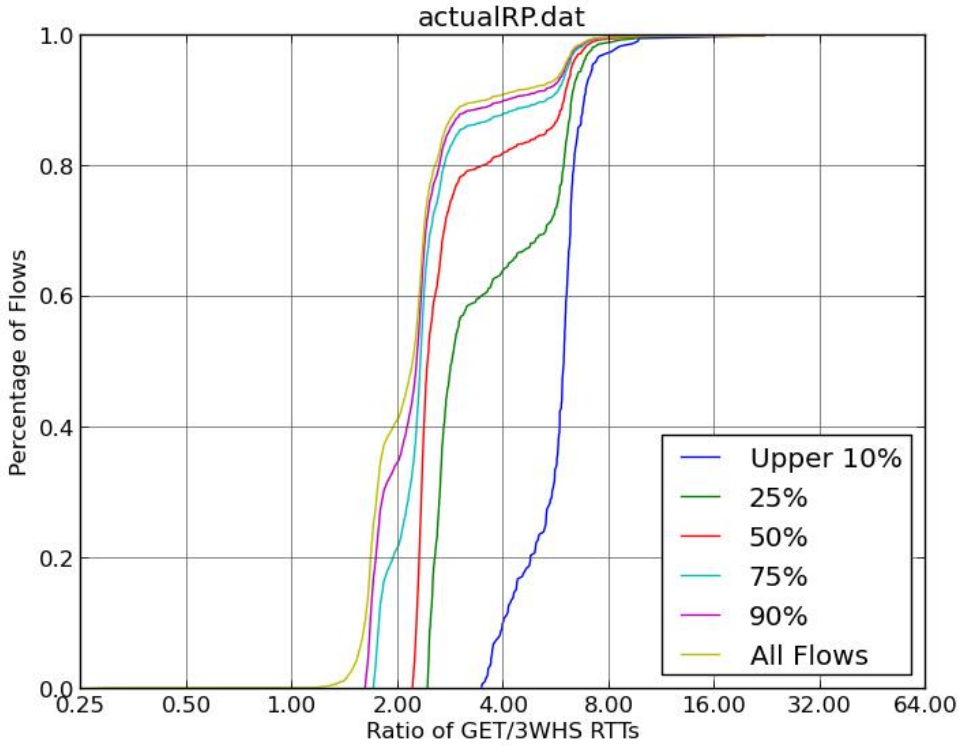


Figure 4.7: Reverse Proxy Websites Cumulative Distribution of Ratios - True Reverse Proxy

Figure 4.7 reflects the RTT ratios for an actual reverse proxy server. We can easily see that the bulk of RTT ratios are roughly 2. Table 4.13 shows the authentic reverse proxy percentile

results. Though our set is limited, we have 2,490 flows against a known reverse proxy server with 100% success. We could set a threshold of 1.6 if we wanted to encompass all possible reverse proxy servers, or set a threshold T between 2 and 3 to discard anomalous RTT ratios. Our value of X needs to be 90% so we encompass the majority of flows while omitting the lowest ratios that may have been affected by caching. On the other hand, abnormally large ratios could be the result of bursty network delays that affect data packet transfers, legacy servers that have limited processing power for data packet transmission when under heavy loads, or possibly packet loss or retransmissions.

Table 4.13: Percentile Values for the Known Reverse Proxy Server

Average Ratio	**Maximum Ratio**	**10th Percentile**	**25th Percentile**	**50th Percentile**	**75th Percentile**	**90th Percentile**
2.42821306276	21.9782352448	3.41394352913	2.42637991905	2.19787526131	1.69977366924	1.60702753067

The comparison of threshold results between Table 4.14 and the other website groups is plain. We have 10 flow sets for the reverse proxy. We will need to create more reverse proxy servers of our own or identify known reverse proxy servers for future work to build upon our threshold identification. We want to use a threshold that is as close to 100% identification against known reverse proxies while not overlapping with known direct connections to maximize true positives and negatives while minimizing false positives and negatives.

Table 4.14: Percent of Reverse Proxy Flow Sets w/in Ratio Thresholds

# Flow Sets with at least X% flows having ratio > T	**Threshold T = 1.25**	**Threshold T = 1.50**	**Threshold T = 1.75**	**Threshold T = 2.00**	**Threshold T = 2.25**
X = 10	10 / 10 = 100%	10 / 10 = 100%	10 / 10 = 100%	6 / 10 = 60%	6 / 10 = 60%
X = 25	10 / 10 = 100%	10 / 10 = 100%	7 / 10 = 70%	6 / 10 = 60%	5 / 10 = 50%
X = 75	10 / 10 = 100%	10 / 10 = 100%	6 / 10 = 60%	6 / 10 = 60%	4 / 10 = 40%
X = 90	10 / 10 = 100%	9 / 10 = 90%	6 / 10 = 60%	5 / 10 = 50%	2 / 10 = 20%

Instead of identifying a statistical tool for identifying the best threshold value, we will attempt to maximize our truth values by using a high threshold. Additionally, we will match

website flow sets across all 10 PlanetLab nodes to identify websites that consistently have RTT ratios above the threshold we set. This will help omit vantage point influence with our fingerprinting. We provide a table in the next subsection to help motivate our choices for the case studies by using the two aforementioned discriminators.

4.4.2 Websites that match Reverse Proxy Threshold

In Table 4.15 we identify those websites within each website group that exhibit high RTT ratios across at least 6 of the 10 flow sets for that website. We look at the upper 90% of flows for each flow set. The domain name and IP address are both provided if available. The optimal fingerprint would have a score of 10/10 with a high threshold. The fingerprint is weaker as the number of 10/10 threshold values in Table 4.15 decreases. We identify the lowest threshold that has 10/10 for a website as a baseline fingerprint because that represents the best performing flow set for the website (i.e., highlighted cells in Table 4.15).

The motivation for desiring 10/10 within a high threshold is based on the expectation that consistently high ratios are a stronger indicator of a reverse proxy. If a particular website shows even a single flow set within a low threshold, we have proof that the target website is capable of servicing fast RTTs for data. The remaining flow sets at higher thresholds then have less value because they could be based on various network delays resulting from vantage point locations.

We want to choose a single website within each set that exhibits behaviors similar to reverse proxies that we can examine further. `Apple.com` did not show strong threshold results in Table 4.15 but we were given information about the server IP address that resulted in interesting values. In the Alexa 5k to 10k group we choose `feedblitz.com` because it has a strong threshold value across all PlanetLab Nodes. `Oreno.co.jp` had the highest threshold match within the Alexa 1 million website group. Lastly, we examine the top three malicious website threshold matches to investigate the possibility of reverse proxies in known malicious domains. We also compared the case study websites against each other to highlight similarities.

Table 4.15: Highest Ratio Websites per Website Group and their Corresponding Fraction of Flow Sets per Ratio Threshold

Website Group	**Domain name / IP Address**	**Threshold T = 1.25**	**Threshold T = 1.50**	**Threshold T = 1.75**	**Threshold T = 2.00**	**Threshold T = 2.25**
Alexa Top 100	apple.com 17.149.160.49	9 / 10	6 / 10	3 / 10	2 / 10	2 / 10
Alexa Top 100	amazon.com 205.251.242.54	9 / 10	7 / 10	5 / 10	3 / 10	3 / 10
Alexa Top 100	flickr.com 68.142.214.24	10 / 10	9 / 10	9 / 10	7 / 10	6 / 10
Alexa Top 100	qq.com 125.39.240.113	10 / 10	10 / 10	6 / 10	1 / 10	1 / 10
Alexa 5k - 10k	feedblitz.com 74.208.13.17	10 / 10	10 / 10	10 / 10	10 / 10	10 / 10
Alexa 5k - 10k	ujian.cc 210.14.128.141	10 / 10	10 / 10	10 / 10	3 / 10	2 / 10
Alexa 5k - 10k	skynewsarabia.com 94.200.220.238	10 / 10	10 / 10	10 / 10	10 / 10	10 / 10
Alexa 5k - 10k	bloggertipstricks.com 162.144.5.3	10 / 10	10 / 10	8 / 10	6 / 10	6 / 10
Alexa 500k - 1m	getrentalcar.com 217.23.2.151	10 / 10	10 / 10	10 / 10	10 / 10	9 / 10
Alexa 500k - 1m	oreno.co.jp 158.199.163.43	10 / 10	10 / 10	10 / 10	10 / 10	10 / 10
Alexa 500k - 1m	accuenergy.com 204.15.197.140	10 / 10	10 / 10	10 / 10	10 / 10	9 / 10
Alexa 500k - 1m	likvidacija-ooo.ru 50.87.150.6	10 / 10	10 / 10	9 / 10	7 / 10	6 / 10
Malicious	(none) 129.121.160.162	10 / 10	10 / 10	10 / 10	10 / 10	10 / 10
Malicious	(none) 149.47.152.172	10 / 10	10 / 10	10 / 10	10 / 10	10 / 10
Malicious	(none) 149.47.155.172	10 / 10	10 / 10	10 / 10	10 / 10	10 / 10
Malicious	(none) 149.47.132.173	10 / 10	10 / 10	10 / 10	10 / 10	9 / 10

4.5 Case Studies

The case studies in this section are a deep dive of the target website results across all 10 PlanetLab nodes. The RTT ratio CDFs, the percentiles, and examination of the actual pcap files are presented to help reinforce our hypothesis and answer some of our research questions from Chapter 1. Each website was chosen because its results were distinct amongst its website group. In the case of the malicious websites, we present three separate case studies since our data set was larger for malicious domains, and these targeted websites are a focus for this research.

4.5.1 Alexa Top 100 Case Study - Apple.com

Apple's website results were chosen for the Alexa top 100 case study because we had access to a reliable source of information regarding how Yahoo is hosted. We used the URL "apple.com" to do a domain name service lookup that resolved the IP address we used (17.149.160.49) to collect our data. Our reliable source of information was that the URL `http://apple.com` was a legacy server, and that the server hosted no content. The server merely redirected traffic to the appropriate Apple webserver based on the type of traffic (i.e., `http://www.apple.com` for HTTP on port 80).

We performed a series of manual inspections on the network traffic generated between the client and the legacy server and the client to `http://apple.com`. We did a DNS lookup to get the IP addresses that are listed for "apple.com" are 17.172.224.47, 17.149.160.49 (the IP address we use to collect network traffic), and 17.178.96.59. The DNS lookup results against `www.apple.com` are different. If a client attempts to connect to `http://apple.com`, the network traffic will show an "HTTP/1.1 301 Moved Permanently" response, redirecting the client to an IP address for one of Apple's web servers. The HTTP 301 response is different from the previously explained HTTP 302 response in that it tells the client to not revisit the URL/IP address in future requests, but it is up to the client whether or not it follows this response. These three legacy servers serve the same web content as `http://www.apple.com` if the web client connects to one of the three legacy server IP addresses in the web browser or uses a command line tool such as wget. The web content is possibly cached at each of these legacy servers for web clients that force HTTP requests to one of the three IP addresses, or the legacy servers are configured as reverse proxy servers.

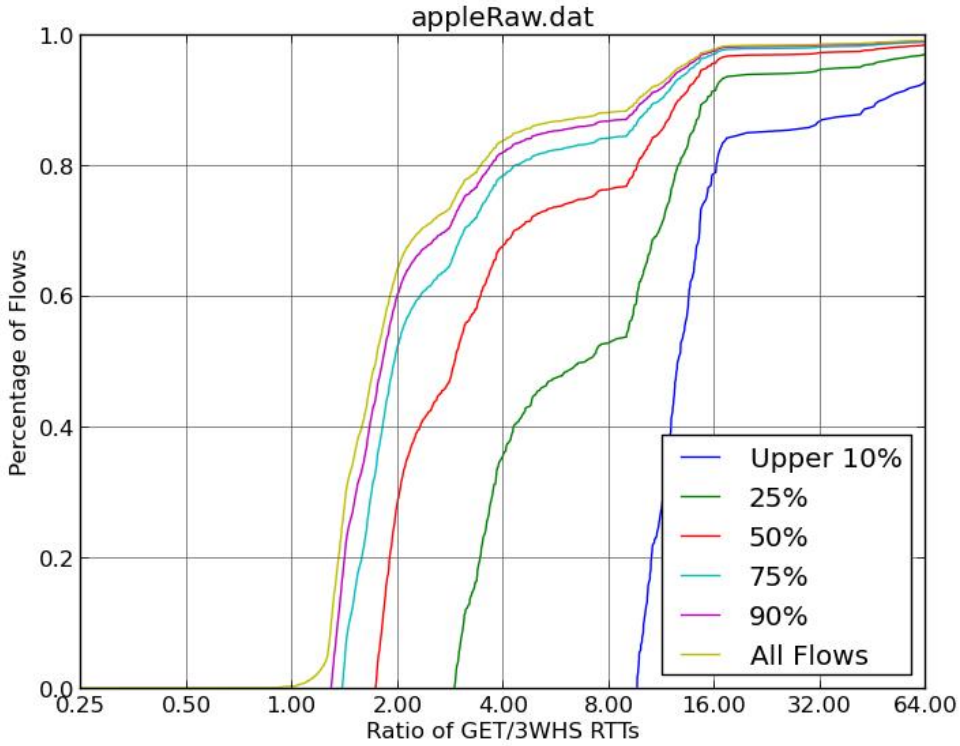


Figure 4.8: Apple.com Cumulative Distribution of Ratios

Table 4.16: Percentile Values for Apple.com

Average Ratio	Maximum Ratio	10th Percentile	25th Percentile	50th Percentile	75th Percentile	90th Percentile
4.20791528898	445.687316895	9.60596847534	2.8955848217	1.72469222546	1.38438487053	1.28809475899

Referring to Table 4.16 and Figure 4.8, we can see that roughly 40% of all flows have a flow set ratio of 1.75 or less. These flow sets are a combination of slower 3WHS RTTs and comparatively fast GET RTTs that result in a lower RTT ratio. These RTT ratios can be misleading in our findings because, though we may have a reverse proxy, spikes in network delay during the 3WHS can cause our ratio results to look like direct connections. When we look at larger percentages of total flows, we can see that at least 80% of all flows have RTT ratios of above 2.0. The legacy servers are slower to respond with data yet still send an HTML 200 OK response, whereas Apple's primary web servers have quick response times with ratios near 1.0 with the

same HTML 200 OK response. The slower response times with matching data retrieval suggests that the URL `http://17.149.160.49` is serving as a reverse proxy when a client directly connects to it.

It is possible that these legacy servers serve as reverse proxies to the actual web servers or other Apple data servers when web clients connect directly to the legacy server IP addresses. Another possibility is that the servers are not built to handle network load for serving content to clients but more suited for redirection as we can see with the DNS resolution tests so the processing delay for data service leads to high RTT ratios, but we believe that the processing delay is a less likely scenario. We assess that processing delay would vary over time based on network load so that the range of ratios would be large, but our ratios are mostly above 1.75 across all vantage points. Our conclusion on this particular case study is that the legacy servers provide redirection when web clients mistakenly type the URL `http://apple.com` into their web browser, but serve web content as reverse proxies when web clients use their network IP addresses directly.

4.5.2 Alexa 5k to 10k - feedblitz.com

We wanted to verify that the URL `http://feedblitz.com` resolved to the same IP address as the URL `http://www.feedblitz.com` because of the findings in the Apple case study. The list of IP addresses returned for both NS lookups contained the IP address we used to generate network traffic (74.208.13.17). Feedblitz advertises itself as an automated marketing service for email and social media [56].

Table 4.17: Percentile Values for feedblitz.com

Average Ratio	Maximum Ratio	10th Percentile	25th Percentile	50th Percentile	75th Percentile	90th Percentile
21.5330613914	93.4940490723	36.8966827393	29.5751972198	22.8499794006	9.2367067337	7.84833526611

The results referenced in Figure 4.9 and Table 4.17 are interesting because the threshold values across all nodes remain high. Furthermore, all flows exhibit RTT ratios of at least 4.0. When we look at the network traffic in a single HTTP request to Feedblitz, it is easy to notice the large time disparity in the 3WHS and GET RTTs. The HTML response for each web request states that the server is running Microsoft IIS/7.0, and states several extra fields reflecting the

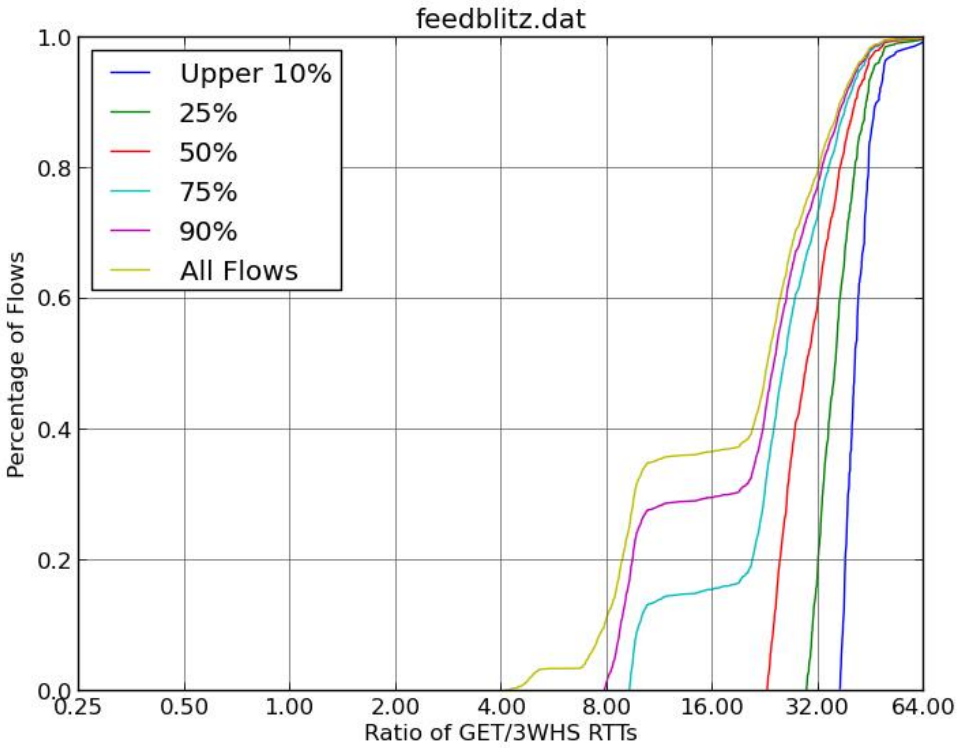


Figure 4.9: feedblitz.com Cumulative Distribution of Ratios

use of ASP.NET and XML-RPC, a remote procedure call using XML over HTTP. Blogging websites use XML RPC and PHP over HTTP to notify externally linked websites that they were referenced within a served page's content. In each of the Alexa website groups, blogging sites regularly had high RTT ratios. It is possible that XML RPC serviced webpages may be set up as reverse proxies.

When we use the WhoIs service to look up the feedblitz IP address, we get information about 1&1 Internet Incorporated, a web hosting service out of Chesterbrook, PA who owns that IP address. They offer a selection of servers for hosting websites, which is likely where our interesting results originate. 1&1 Internet Inc. offer three types of servers; virtual servers, dynamic cloud servers, and dedicated servers [57]. The delays we observe in getting the data is likely because of an Internet-facing web server used by 1&1 Internet Inc. to proxy their web services.

The connection to 1&1's Internet-facing server is quick, but when we request `feedblitz.com`, the server is required to find the appropriate data on whichever server `feedblitz.com` is utiliz-

ing, obtain the data, and forward the data from one of their internal servers on to the requesting client. We assess it is unlikely that `feedblitz.com` is using a dedicated server because the response times would be significantly faster. `feedblitz.com` is likely using a virtual or dynamic cloud server which is why we see such a large delay in data response. The data timing delays are consistent with reverse proxies, and we therefore assess that `feedblitz.com` is being hosted behind a reverse proxy at 1&1 Internet Incorporated.

4.5.3 Alexa 500k to 1 Million - oreno.co.jp

`Oreno.co.jp` is a Japanese restaurant website. It uses XML RPC and PHP web pages similar to Feedblitz in the Alexa 5k to 10k case study. An interesting phenomenon with this website occurs when requesting the website via the domain name in a web browser or HTTP request tool (i.e., wget). The web traffic to `http://oreno.co.jp` shows the IP address as 158.199.163.43, but returns an "HTTP/1.1 301 Moved Permanently" response. The response redirects the client to the exact same IP address and TCP port with a separate domain name (`http://www.oreno.co.jp`). If we attempt to connect directly to the IP address, we do not receive the HTTP 301 response. We collected all of our network traffic by connecting directly to the IP address so the HTTP 301 response did not impact our results.

The strange behavior with the domain name responses may be indicative of a reverse proxy server because the domain name service would resolve both domain names (with or without the "www") to the same IP address. A reverse proxy may be configured to only retrieve web content for a specific HTTP header that includes the "www" in the request. When we use the IP address as a hard-coded destination, the domain name service is bypassed so we do not run into the same problem.

There were similar RTT ratios for this website compared to Feedblitz. We had an extreme case with our Singapore PlanetLab node that had RTT ratios greater than 198 for 90% of all flows. If we omit this particular node, we still see consistently high RTT ratios across the remaining nodes that are at least 2.77 for 90% of all flows. The website uses javascript, cascading style sheets, XML, and PHP. The WhoIs results for `http://oreno.co.jp` fall into the ownership of KDDI Web Communications Incorporated, a web-hosting service out of Japan [58]. We expect that the configuration of KDDI's Internet-facing IP address is similar to that of 1&1 Internet Inc. for `feedblitz.com`. Though KDDI does not advertise server types on their website, they do mention the high need for Internet domains for Japanese businesses but with a low supply. They offer their hosting services as a solution. If they are hosting over 40,000 corporate

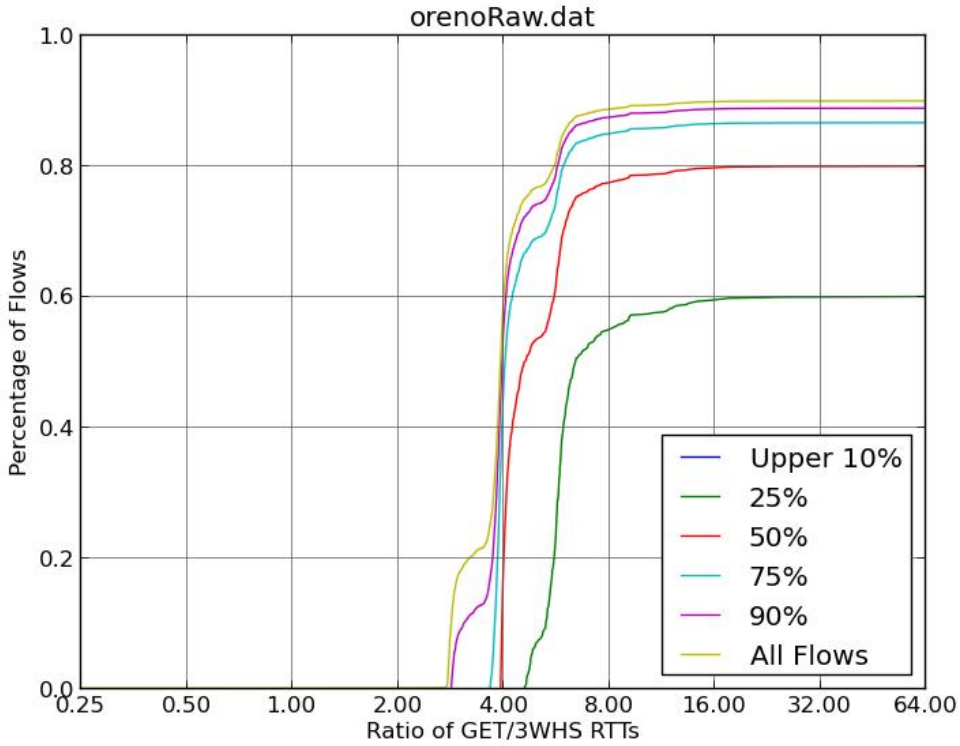


Figure 4.10: Oreno.co.jp Cumulative Distribution of Ratios

clients [58], it is likely they have a large set of data centers to store all of the content. Clients will easily connect to the Internet facing web server, but then have to wait for the data to get transferred from an internal data server back out to the client. The delay is that of a reverse proxy.

Table 4.18: Percentile Values for Oreno.co.jp

Average Ratio	Maximum Ratio	10th Percentile	25th Percentile	50th Percentile	75th Percentile	90th Percentile
27.4278299519	283.344116211	117.103462219	4.58236169815	3.92079305649	3.65683436394	2.99382328987

We believe `http://oreno.co.jp` and KDDI Web Communications showed definitive signs of reverse proxy characteristics. The consistently high RTT ratios are the strongest indicator. The larger then Internet becomes, the more web hosting services must utilize the larger range of domain names to make up for the reduced IP address space. DNS allows web hosting services like KDDI to utilize internal data storage and reverse proxy servers to handle web requests.

4.5.4 Malicious Website One - 129.121.160.162

Malicious website one (129.121.160.162) is more specifically referenced at MalwareDomainList.com [52] as a Blackhole exploit kit. The website is hosted in the United States by Oso Grande Technologies in Albuquerque, New Mexico. The blackhole exploit uses obfuscated javascript within a webpage to identify features of the client's machine then compromises the machine based on the results. The webpages are most likely used for short periods of time under the expectation that the webpages will eventually be shut down once they are identified as malicious. Blackhole exploit kits have Russian origins. The kits made up 41% of web attacks in 2012 as reported by the Symantec Corporation [59].

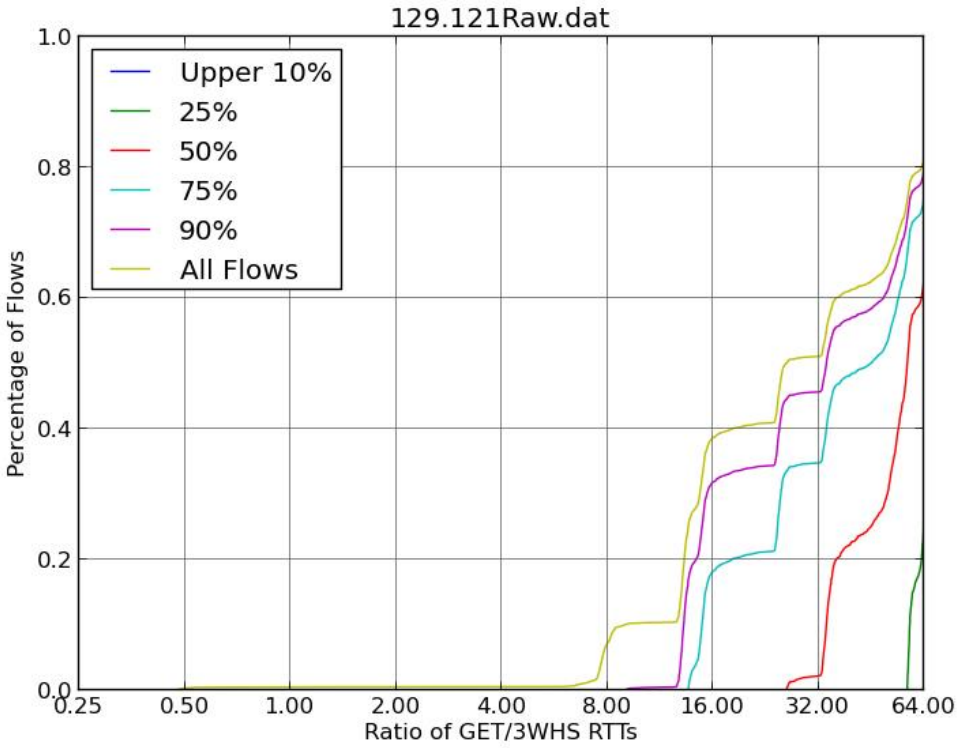


Figure 4.11: 129.121.160.162 Cumulative Distribution of Ratios

Referencing Figure 4.11 and Table 4.19, we can see that the RTT ratios are consistently high across all PlanetLab nodes. When we visit the website, we get a webpage that looks like a simple blog focused on data merging and smoking cigarettes. Shortly after the page loads, an advertisement pops up for cigarette sales. The link for the advertisement goes to `theecig.net/v2savemore`. MalwareDomainList.com [52] provides the IP address as a malicious website, but the URL for the malicious exploit is at

`http://129.121.160.162/f2bca24cbac8843a780d35653bfe8548/q.php`. The aforementioned page no longer exists, so we are not able to run the exploit in a sandboxed environment. Based on the timing results, we assess that this website is behind a reverse proxy.

Table 4.19: Percentile Values for 129.121.160.162

Average Ratio	Maximum Ratio	10th Percentile	25th Percentile	50th Percentile	75th Percentile	90th Percentile
39.1598096188	670.701721191	69.3232040405	57.3231506348	25.9222316742	13.6023540497	8.92593288422

4.5.5 Malicious Website Two - 149.47.152.172

Malicious website two (149.47.152.172) is also referenced as a Blackhole exploit kit, but it is under version 2.0 [52]. The website is hosted by Precipice Webhosting services out of New York City, New York. We can see in Table 4.20 that the RTT ratios for this website are much lower than that of malicious website one (Table 4.19). The RTT ratios are consistently above 3.0 across all PlanetLab nodes, with the lowest around 2.2 for 90% of all flows across all nodes.

We were unable to make connections to either the IP address directly or the URL referenced as malicious at MalwareDomainList.com [52]. The malicious URL string is listed as `http://149.47.152.172/Web/links/replacement-based_destroy-varies.php`. We could not conduct any further analysis of the actual network traffic to the site beyond what our initial traffic depicted. The RTT ratios suggeste that this is also a reverse proxy as the RTT ratios were consistently high across all PlanetLab nodes.

Table 4.20: Percentile Values for 149.47.152.172

Average Ratio	Maximum Ratio	10th Percentile	25th Percentile	50th Percentile	75th Percentile	90th Percentile
8.38564520167	30.7004337311	16.1830101013	12.2755508423	6.76642513275	3.60729312897	3.35419392586

4.5.6 Malicious Website Three - 149.47.155.172

Malicious website three (149.47.155.172) exhibits the same behaviors as malicious website two. Furthermore, it is reported to be using the same exploit (Blackhole Tool Kit 2.0) [52]. The

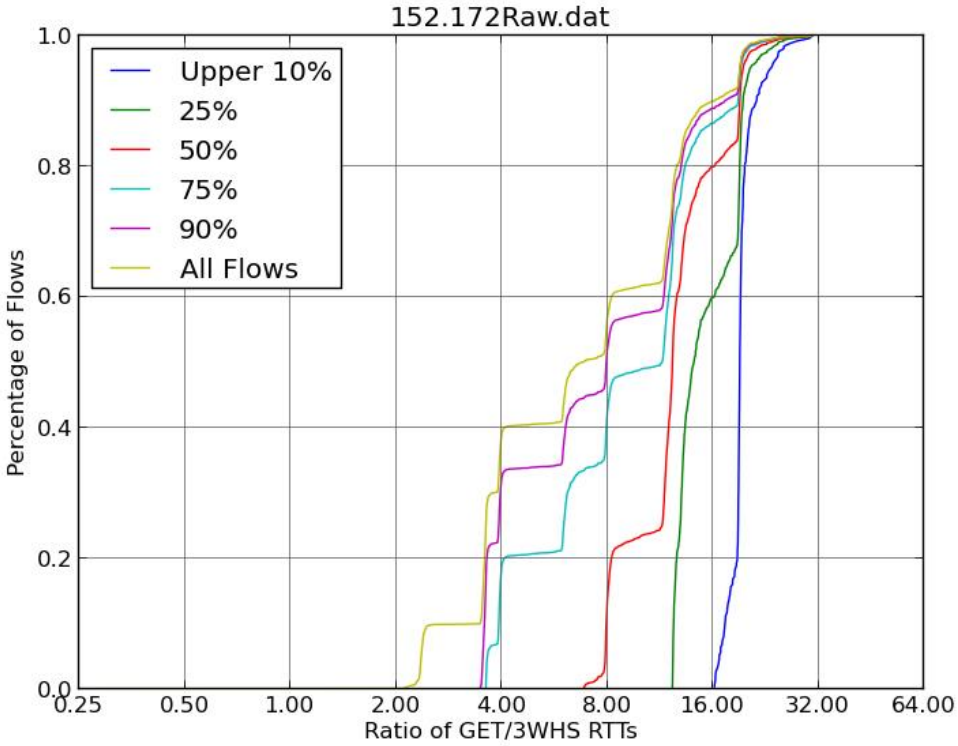


Figure 4.12: 149.47.152.172 Cumulative Distribution of Ratios

HTML returned from the network traffic includes the URL `http://vitashoesco.com/`. We attempted to connect to this website, the IP address we got from Malware Domain List, and the URL reported as an exploit. None of the URLs were still active. In the same fashion as the previous two malicious sites that showed similar behavior, we reference Figure 4.13 and Table 4.21 as evidence that the server was using a reverse proxy to hide the backend malicious behaviors.

Table 4.21: Percentile Values for 149.47.155.172

Average Ratio	Maximum Ratio	10th Percentile	25th Percentile	50th Percentile	75th Percentile	90th Percentile
8.79291193862	111.13899231	17.8608493805	12.6154546738	7.89357471466	3.61290693283	2.99382328987

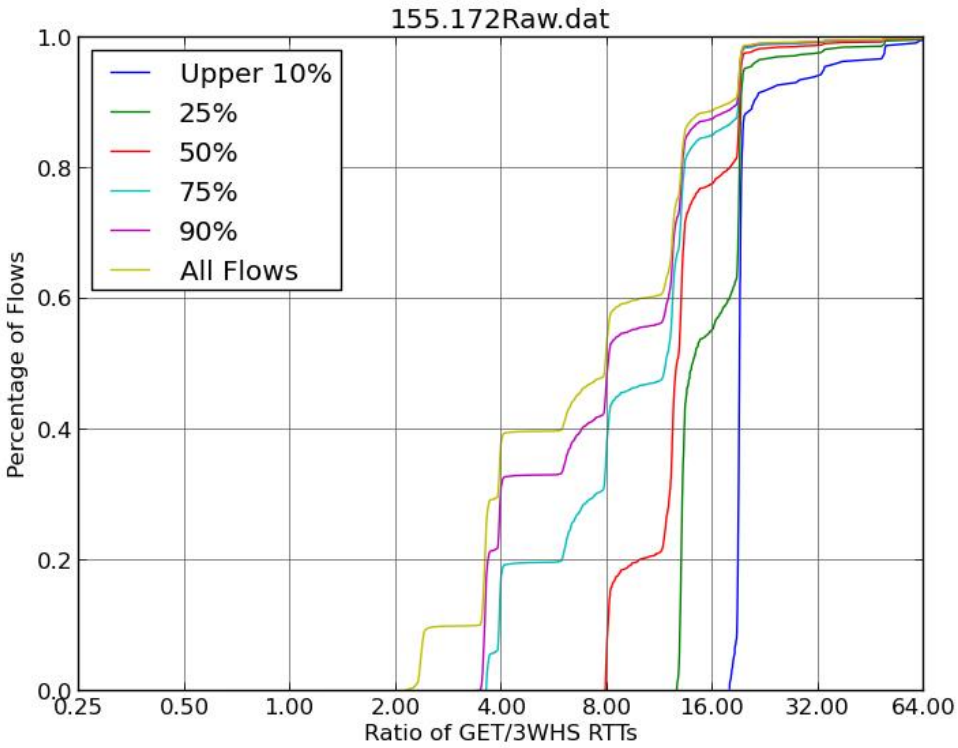


Figure 4.13: 149.47.155.172 Cumulative Distribution of Ratios

CHAPTER 5:
CONCLUSIONS AND FUTURE WORK

We present our conclusions in this chapter based on the results and analysis against Alexa website groups and malicious websites. We identify the strengths and research contributions of our experiments as well as the shortfalls. These shortfalls serve as motivation to improve our established methodology in future work.

5.1 Conclusions

We developed a methodology to actively probe targeted websites across a set of vantage points, gather the network traffic from the HTTP requests, and analyze the RTT results from HTTP sessions to identify reverse proxies. The vantage points were globally distributed to check RTT consistency and increase the set of network traffic we could analyze. We looked at three sets of selected websites using Alexa's website ranking system [50]. The three sets covered the most popular sites, mid-grade popularity sites such as educational institutions and entertainment providers, and lower popularity sites such as foreign websites, small business e-commerce, and web blogs. We also gathered a large set of flows against known malicious websites using a database that was current [52].

Our analysis was based on the ratio of the GET RTT over the 3WHS RTT. This ratio gave us a simple value that we could compare against our known set of reverse proxy servers. We established a range of threshold values based on the known reverse proxy ratios that we then compared across all website groups. We discovered that large RTT ratios existed in all types of website groups. Approximately 15-20% of all website groups that we analyzed showed reverse proxy-like timing ratios. The ratio marker indicates that data-centric delays in HTTP connections apply to high performance websites, blogs, and even malicious websites.

We claim that our ratio marker could be caused by several factors. Processing delays in the lower performance websites could cause slow response times for data when under heavy traffic loads. Medium-level popularity websites are likely hosted behind CDNs, web-hosting services, or possibly reverse proxies of their own. The most popular websites likely use CDN technology and their own web-hosting servers that may include reverse proxies. Malicious websites are likely using compromised web-hosting services or home workstations with built-in reverse proxies to hide their activities, and the larger RTT ratios could be a combination of

an aforementioned factor with processing delays because of network edge web hosting.

The Alexa Top 100 website group showed high performance results given that the majority of RTT ratios were under the lowest ratio threshold we established (1.25). The Alexa 5k to 10k website group had a larger disparity between the low ratios and the upper ratios (See Fig. 4.3) when compared to Alexa's top 100 website group. Similarly, Alexa's 500k to 1 million website group had roughly 75% of all flows with a ratio of 1.25 or less. The malicious websites were the closest in comparison to Alexa's 500k to 1 million group, exhibiting consistently low ratios except that the difference between the averages and the upper 10% of ratios was large, hence the long-tailed CDF.

Without having a way to verify our results, we cannot say with 100% certainty that our assessed reverse proxies are, in fact, reverse proxies in reality. However, the test cases we performed using manual packet inspection all suggest our methodology is highly accurate. We did see some interesting results in our case studies. `Apple.com` and `oreno.co.jp` provide redirection when "www" is not included in the URL. Apple redirected to a different server, though if a client connected to the IP address of `apple.com` the client would still get the same website as `www.apple.com`. `Oreno.co.jp` would redirect the client to the same IP address and the same port, which seems like a misconfigured webserver but could be indicative of a reverse proxy.

5.1.1 Strengths and Contributions

The greatest contribution of our research is the application layer fingerprinting methodology. We discovered that HTTP could be used to reveal indistinct characteristics of servers using the timing of each packet in an HTTP request. The time required for a TCP session to be established is roughly equivalent for most HTTP requests and their corresponding responses. We showed that a ratio of the GET RTT over the 3WHS RTT from a TCP session could be used with known timing thresholds of a reverse proxy to identify the server type. The methodology we created can be used from any client to any server that supports HTTP traffic. The modularity and simplicity of our approach can be correlated with other known fingerprinting techniques to have higher levels of confidence in networking signatures as well.

Secondly, we performed a number of small experiments throughout this research to carefully validate our methodology and expectations. We built a series of small scale tests using servers we controlled in order to manually inspect packet dynamics from the client and the server van-

tage points. Developing timing diagrams based on the resulting network traffic helped us realize that Karn's algorithm would not be sufficient, which forced us to discard popular traffic analysis tools widely used on the Internet. Furthermore, the small experiments and class projects gave us helpful experiences that led us to approach data collection and organization in more efficient ways.

The frequent use of scripting helped us automate mundane tasks such as data organization, data collection, graphing, and table building. We were able to quickly create scripts using bash and/or python to meet most of our needs. We had a few scripts that were more robust, such as the data collection and graphing scripts. We were able to customize these scripts using parameters or some manual changes to the syntax to automate valuable steps in our research. Python's matplotlib and numpy modules were significant in helping us graph CDFs across large data sets in our python scripts, and saved us a lot of manual effort.

The number of HTTP requests submitted per website across each node gave us sufficiently large data sets. We believe our data sets were big enough to outweigh most anomalous behavior in network delays that could have swayed our results. The number of requests could be increased, but the results would likely remain consistent. In fact, we could have likely performed 100 or fewer HTTP requests instead of 250 and still gained the same insights that we did in our research.

5.1.2 Shortfalls

The number of known proxy servers is the most glaring shortfall in our research. We executed HTTP requests across hundreds of popular and malicious websites but had only one known reverse proxy server by which to compare. We were able to make assessments on fingerprinted websites but the training set must be larger in future work. This is imperative so that an accurate threshold can be set.

Domains resolve to different IP addresses depending on the vantage point, which can cause inconsistencies in RTT comparisons across global vantage points. To work around this issue, we chose a single vantage point (Monterey, CA) to pull IP addresses. These IP addresses were used to execute the HTTP requests instead of domain names. DNS lookups can return CDN servers, legacy servers, false destinations, or fail altogether depending on the input. When we searched for the IP address of Apple.com, we searched `apple.com` instead of `www.apple.com`. The former provides a group of IP addresses to legacy servers. The latter returns a group of IP

addresses to current websites for the Apple corporation.

The IP address lists in future work should be inspected thoroughly and then put into subgroups. An example could be the websites in the Alexa top 100. If a website IP address points to a CDN server such as Akamai, it is put into a "known CDN" subgroup. Alternatively, the website IP address could be put into "known direct connection" or "unknown" as well. This would help with data organization and analysis.

5.2 Future Work

Several research questions we wished to answer circled around building a machine learning classification mechanism, which we did not accomplish in our work. We are currently working on a classifiication solution for future projects. We were unable to answer the following research questions in this thesis:

1. Given a set of non-proxy webservers, their IP addresses, a set of known reverse proxy webservers and their IP addresses, can an automated Internet request program be trained to classify unknown servers as non-proxy webservers or reverse proxy webservers? If so, what is the confidence interval for classifying unknown servers with the trained program?
2. What is the false positive rate and false negative rate for the classifier results? The F-Score?
3. Is the timing data equivalent regardless of the HTTP request format (GET, POST, etc.)?
4. How does caching affect reverse proxy RTTs and how can we detect caching?
5. What is the time granularity required to differentiate 3WHS RTTs from GET RTTs? Are the current clock resolutions sufficient?
6. Can we differentiate between reverse proxy servers and CDNs with a classifier? What level of precision do we need in our threshold to reduce false positives and false negatives concurrently?

We recommend testing this approach against web servers that have system administrators who can be contacted to verify the presence of a reverse proxy. Our method was not verified for any of our research aside of one website which makes proving our methodology's accuracy and precision difficult to quantify. Additionally, more global vantage points should be used to check domain names to IP addresses, increase data collection for more comparative analysis, and widen the scope of possible RTT ratios. The vantage points are affected by some delays more than others based on distance to the webserver, network path, network infrastructure, and

so forth. As the number of vantage points increases, so does the confidence in fingerprinting websites because the RTT ratios can be calculated using a diverse set of delays. We used mostly IP addresses that were located in the United States. We want to use a much larger variety of vantage points in future testing, specifically spreading our selection of vantage points around the globe to maximize the diversity of network delays that affect timing. Additionally, the results may be more varied if we performend DNS lookups from a Russian or Chinese vantage point. Though our data set was large enough for thesis research, a larger set of flows with more delineated website groups could be easier to analyze as well.

A classifier that can perform machine learning techniques against websites is the end goal. We would like to refine our code base to train against known reverse proxies and direct connections to build statistically accurate thresholds then classify websites. A third server group could be CDNs, whose ratios may fall in between the timing thresholds of reverse proxies and direct connections. We could then identify the true statistical value of our approach, identify weaknesses, and fix those weaknesses or fill the gap with another fingerprinting technique to strengthen the overall classifier.

THIS PAGE INTENTIONALLY LEFT BLANK

REFERENCES

[1] Symantec Security Company, "Internet security threat report - 2011 threat landscape," April 2012. [Online]. Available: https://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_2011_21239364.en-us.pdf

[2] J. Crist, "Web-based attacks," *SANS Institute InfoSec Reading Room*, 2007.

[3] G. Sinclair, C. Nunnery, and B.-H. Kang, "The waledac protocol: The how and why," in *Malicious and Unwanted Software*, 2009, pp. 69–77.

[4] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna, "Your botnet is my botnet: Analysis of a botnet takeover," in *Computer and Communications Security*, ser. CCS '09. New York City, NY: ACM, 2009, pp. 635–647.

[5] D. Barroso, "Botnets-the silent threat," in *ENISA Position Papers*, ser. S21sec, no. 3. Crete, Greece: ENISA, November 2007.

[6] L. Wang, F. Douglis, and M. Rabinovich, "Forwarding requests among reverse proxies," in *International Web Caching and Content Delivery Workshop*, ser. IWCW '00, no. 5. Princeton, NJ: Princeton University Press, 2000.

[7] G. Barish and K. Obraczke, "World wide web caching: Trends and techniques," *Communications Magazine*, vol. 38, no. 5, pp. 178–184, 2000.

[8] G. M. Snow, "Statement before the senate judiciary committee, subcommittee on crime and terrorism," April 2011. [Online]. Available: http://www.fbi.gov/news/testimony/cybersecurity-responding-to-the-threat-of-cyber-crime-and-terrorism

[9] G. Hogben, D. Plohmann, E. Gerhards-Padilla, and F. Leder, "Botnets: Detection, measurement, disinfection and defence," *European Network and Information Security Agency*, 2011.

[10] K. B. Knight, J. Lawwill Jr., and B. L. Pulito, "Detecting a reverse proxy and establishing a tunneled connection therethrough," September 2003, U.S. Patent 7,272,642.

[11] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. Wolff, "A brief history of the internet," *Computer Communication Review*, vol. 39, no. 5, pp. 22–31, 2009.

[12] L. Nolan, R. Beverly, and G. Xie, "Transport traffic analysis for abusive infrastructure characterization," Master's thesis, Monterey, CA. Naval Postgraduate School, 2012.

[13] Farlex, Inc., "The free dictionary: Definition of proxy," 2009. [Online]. Available: http://www.thefreedictionary.com/proxy

[14] Open Directory Project, "Wikipedia: Proxy server," June 2013. [Online]. Available: https://en.wikipedia.org/wiki/Proxy_server

[15] J. F. Kurose and K. W. Ross, *Computer Networking: A Top Down Approach*. Addison Wesley, 2001, vol. 2.

[16] B. D. Davison, "A web caching primer," *Internet Computing*, vol. 5, no. 4, pp. 38–45, 2001.

[17] A. Stricek, "A reverse proxy is a proxy by any other name," *SANS Institute InfoSec Reading Room*, vol. 13, 2002.

[18] I. Sysoev, "Nginx: Free, open-source http server and proxy," May 2013. [Online]. Available: http://wiki.nginx.org/Main

[19] The Apache Group, "Apache - http server project," May 2013. [Online]. Available: https://httpd.apache.org/

[20] D. Wessels, "Squid caching proxy," June 2013. [Online]. Available: http://www.squid-cache.org/

[21] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616 (Draft Standard), Internet Engineering Task Force, Jun 1999, updated by RFCs 2817, 5785, 6266, 6585. [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt

[22] Akamai Content Delivery Service, "Akamai media streaming glossary: Content delivery network (cdn)," July 2013. [Online]. Available: http://www.akamai.com/html/solutions/sola_cdn.html

[23] ——, "Akamai customer list," July 2013. [Online]. Available: http://www.akamai.com/html/customers/customer_list.html

[24] T. Kohno, A. Broido, and K. C. Claffy, "Remote physical device fingerprinting," *Dependable and Secure Computing*, vol. 2, no. 2, pp. 93–108, 2005.

[25] V. Jacobson, "tcpdump - live network capture," June 2012. [Online]. Available: http://www.tcpdump.org

[26] V. Paxson, "On calibrating measurements of packet transit times," in *Performance Evaluation Review*, vol. 26, no. 1. New York City, NY: ACM, 1998, pp. 11–21.

[27] A. Hintz, "Fingerprinting websites using traffic analysis," in *Privacy Enhancing Technologies*, ser. PET '02. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 171–178.

[28] X. Gong, N. Kiyavash, and N. Borisov, "Fingerprinting websites using remote traffic analysis," in *Computer and Communications Security*, ser. CCS '10. New York City, NY: ACM, 2010, pp. 684–686.

[29] D. Herrmann, R. Wendolsky, and H. Federrath, "Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier," in *Cloud Computing Security*, ser. CCSW '09. New York City, NY: ACM, 2009, pp. 31–42.

[30] J. Zhang, R. Perdisci, W. Lee, U. Sarfraz, and X. Luo, "Detecting stealthy p2p botnets using statistical traffic fingerprints," in *Dependable Systems & Networks*, ser. DSN '11. Washington, DC: IEEE Computer Society, 2011, pp. 121–132.

[31] B. Shebaro, F. Perez-Gonzalez, and J. R. Crandall, "Leaving timing-channel fingerprints in hidden service log files," *Digital Investigation*, vol. 7, pp. S104–S113, 2010.

[32] G. Danezis and R. Clayton, "Introducing traffic analysis," Boca Raton, FL, pp. 95–117, 2007.

[33] V. Paxson, "End-to-end internet packet dynamics," in *Computer Communication Review*, vol. 27, no. 4. New York City, NY: ACM SIGCOMM, 1997, pp. 139–152.

[34] H. Martin, A. McGregor, and J. Cleary, "Analysis of Internet delay times," vol. 1. Hamilton, NZ: Springer-Verlag, 2000, p. 2.

[35] P. Karn and C. Partridge, "Improving round-trip time estimates in reliable transport protocols," *Computer Communication Review*, vol. 17, no. 5, pp. 2–7, 1987.

[36] J. Aikat, J. Kaur, F. D. Smith, and K. Jeffay, "Variability in tcp round-trip times," in *Conference on Internet Measurement*, vol. 27, no. 29, Miami, FL, 2003, pp. 279–284.

[37] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," RFC 2581 (Proposed Standard), Internet Engineering Task Force, Apr 1999, obsoleted by RFC 5681, updated by RFC 3390. [Online]. Available: http://www.ietf.org/rfc/rfc2581.txt

[38] J. Micheel, S. Donnelly, and I. Graham, "Precision timestamping of network packets," in *Workshop on Internet Measurement*. New York City, NY: ACM SIGCOMM, 2001, pp. 273–277.

[39] WAND Research Group: University of Waikato, "Dag network cards," June 2013. [Online]. Available: http://dag.cs.waikato.ac.nz/

[40] GoDaddy Corporate, "Godaddy web-hosting services," 2013. [Online]. Available: http://www.godaddy.com

[41] G. Combs, "Wireshark - network protocol analyzer," August 2013. [Online]. Available: http://www.wireshark.org

[42] J. Postel, "Transmission Control Protocol," RFC 793 (Internet Standard), Internet Engineering Task Force, Sep 1981, updated by RFCs 1122, 3168, 6093, 6528. [Online]. Available: http://www.ietf.org/rfc/rfc793.txt

[43] V. Jacobson, R. Braden, and L. Zhang, "TCP Extension for High-Speed Paths," RFC 1185 (Experimental), Internet Engineering Task Force, Oct 1990, obsoleted by RFC 1323. [Online]. Available: http://www.ietf.org/rfc/rfc1185.txt

[44] V. Jacobson and R. Braden, "TCP extensions for long-delay paths," RFC 1072 (Historic), Internet Engineering Task Force, Oct 1988, obsoleted by RFCs 1323, 2018, 6247. [Online]. Available: http://www.ietf.org/rfc/rfc1072.txt

[45] S. Ostermann, "tcptrace - tcpdump analysis tool," November 2003. [Online]. Available: www.tcptrace.org

[46] M. Mellia and M. Munafo, "Tstat - tcp statistic and analysis tool," April 2012. [Online]. Available: http://tstat.tlc.polito.it/index.shtml

[47] H. P. Pfeifer, “Captcp - tcp analyzer for pcap files,” February 2013. [Online]. Available: http://research.protocollabs.com/captcp/

[48] Internet Assigned Numbers Authority, “Service name and transport protocol port number registry,” August 2013. [Online]. Available: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

[49] Planetlab Consortium, “Planetlab global research network,” 2007. [Online]. Available: http://www.planet-lab.org

[50] Alexa Top Websites, “The top 500 sites on the web (global).” March 2013. [Online]. Available: http://www.alexa.com/topsites

[51] Amazon.com, Inc., “Amazon.com,” August 2013. [Online]. Available: http://www.amazon.com/

[52] S. Burn, “Malware domain list,” August 2013. [Online]. Available: http://www.malwaredomainlist.com/

[53] Precipice, Inc., “Precipice web hosting services,” 2013. [Online]. Available: http://www.precipiceinc.com/

[54] CleanMX, “Clean mx realtime database - public access query for phishing url,” August 2013. [Online]. Available: http://support.clean-mx.de/clean-mx/phishing.php?ip=149.47.132.252\&sort=response\%20desc

[55] TorrentCity, “Torrent proxies - pirate bay proxy list.” [Online]. Available: http://www.torrentproxies.com

[56] P. Hollows, “Feedblitz: The all-in-one solution for email & social media marketing automation,” 2005. [Online]. Available: http://www.feedblitz.com

[57] R. Hoffman, “1 & 1 internet incorporated - web hosting services,” 2013. [Online]. Available: http://1and1.com/

[58] KDDI Group, “KDDI web communications,” 1997. [Online]. Available: http://www.kddi-webcommunications.co.jp/english/

[59] H. Suri, “Symantec official blog: The blackhole theory,” February 2011. [Online]. Available: http://www.symantec.com/connect/blogs/blackhole-theory

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia
2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California